VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

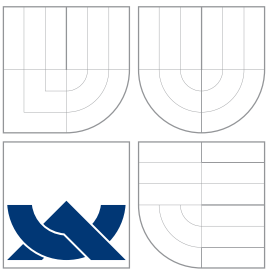
OPTIMALIZACE APLIKACE TESTU ČÍSLICOVÝCH SYSTÉMŮ PRO NÍZKÝ PŘÍKON

DIZERTAČNÍ PRÁCE
PHD THESIS

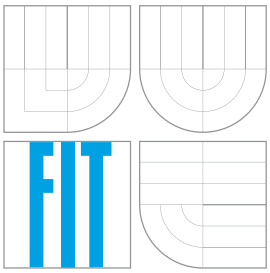
AUTOR PRÁCE
AUTHOR

Ing. JAROSLAV ŠKARVADA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE APLIKACE TESTU ČÍSLICOVÝCH SYSTÉMŮ PRO NÍZKÝ PŘÍKON

DIGITAL SYSTEMS TEST APPLICATION OPTIMIZATION FOR LOW POWER CONSUMPTION

DIZERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE — Ing. JAROSLAV ŠKARVADA
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. ZDENĚK KOTÁSEK, CSc.
SUPERVISOR

BRNO 2009

Abstrakt

Práce se zabývá redukcí příkonu číslicového obvodu během aplikace testu. Zvýšený příkon obvodu během aplikace testu je problematický zejména u VLSI obvodů, protože se mohou projevovat negativní jevy jako je např. pokles napětí na napájecích linkách, přerušení spojů v důsledku elektromigrace, rušení způsobené elektromagnetickou indukcí a zvýšení teploty čipu. V případě, že implementace obvodu není dostatečně dimenzována na tento příkon, mohou být negativně ovlivněny výsledky testu. Dimenzování obvodu pro účely testu je nákladné a neefektivní, protože většinu svojí životnosti bude obvod provozován v normálním funkčním režim činnosti. Z tohoto důvodu bývají aplikovány různé optimalizační metody, jejichž cílem je dosáhnout redukce příkonu obvodu během aplikace testu. Většina těchto metod redukuje příkon prodloužením aplikace testu, takže nedochází k úspoře odebrané energie. Existují však i metody redukující odebranou energii, což umožňuje např. snížit náklady na testování obvodů při výrobě.

V práci jsou analyzovány příčiny zvýšeného příkonu obvodu během aplikace testu při srovnání s běžným funkčním režimem činnosti. Jsou popsány existující metody pro redukci dynamické a statické složky příkonu obvodu během aplikace testu. Je navržena a algoritmicky popsána metoda založená na principu současné optimalizace pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce scan. Metoda je primárně určena pro redukci dynamické složky příkonu u obvodů obsahujících plný řetězec scan, avšak je úspěšně použitelná i pro kombinační obvody bez řetězce scan. U těchto obvodů ovšem dosahuje nižší redukce příkonu. Metoda umožňuje nejen redukovat dynamickou složku příkonu během aplikace testu, ale umožňuje i redukovat energii odebranou ze zdroje. Metoda využívá pro prohledávání rozsáhlého stavového prostoru úlohy genetického algoritmu. Pro stanovení příkonu obvodu se využívá simulace testu s využitím standardní technologické knihovny, což umožňuje získat přesnější výsledky ve srovnání s jednoduššími metodami založenými např. na výpočtu Hammingovy vzdálenosti mezi jednotlivými testovacími vektory. V rámci řešení práce byla metoda implementována a aplikována na existující benchmarkové obvody. V práci jsou též prezentovány dosažené výsledky včetně porovnání s existujícími metodami.

Klíčová slova

Číslicový obvod, testovací vektory, scan registry, optimalizace testu, redukce příkonu.

Abstract

The thesis deals with power consumption reduction of digital circuit during test application. Higher power consumption during the test application is problematic especially for VLSI circuits due to negative effects such as voltage drops on supply lines, electromigration resulting in connections cut, electromagnetic induction resulting in signals interference and overheating. In cases when the chip is not well designed for a such power consumption the test results can be affected. Designing the chip for the power consumption during the test application is expensive and non effective, because most of the time the chip will operate in normal functional mode and the design will be uselessly oversized. That is why the optimization methods for power consumption reduction during the test application were introduced. Generally the application of most of these methods prolongs the test such that the energy consumption is not reduced. But there also exist such specialized methods that allows not only the power reduction but also the energy consumption to be reduced. Such methods allow, for instance, cost reduction during the mass production of digital circuits due to energy savings at check out testing.

The thesis presents the analysis of sources of increased power consumption during the test application in comparison with normal functional mode of operation. Up to date methods for reduction of dynamic and static power during the test application are also described. Next the methodology for simultaneous reordering of test vectors and scan registers is proposed and described by algorithms. The methodology is primarily used for reduction of dynamic part of power for full scan based circuits, but can be used as well for combinatorial circuits only with the little drop in power reduction. The methodology also reduces the energy consumption during the test application. For the browsing of the large solution space of the task the genetic algorithm is used. For fitness calculation the power consumption value is used. The power consumption is evaluated by simulation of test application over the technological library. This approach allows to obtain more precise results in comparison with simple methods (e.g. methods based on computation of Hamming distance between test vectors). The method was implemented and applied on established benchmark sets. The results gained from experiments over these benchmarks and comparison with existing methods is also presented.

Keywords

Digital circuit, test vectors, scan registers, test optimization, power consumption reduction.

Citace

Jaroslav Škarvada: Optimalizace aplikace testu číslicových systémů pro nízký příkon, dizertační práce, Brno, FIT VUT v Brně, 2009

Optimalizace aplikace testu číslicových systémů pro nízký příkon

Prohlášení

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

.....
Jaroslav Škarvada
17. července 2009

Poděkování

Děkuji svému školiteli docentu Zdeňku Kotáskovi za skvělé vedení během mého studia, za cenné rady, připomínky a komentáře k mé práci. Dále děkuji svým rodičům za finanční i morální podporu během mého studia, bez níž by tato práce nemohla vzniknout. Tato práce byla podporována grantovou agenturou GAČR v rámci grantu GA102/04/0737 – Moderní metody syntézy číslicových obvodů a agenturou FRVŠ v rámci grantů FR3383/2006/G1 – Plánování testu vestavěných systémů zohledňující příkon elektrické energie a FR2472/2007/G1 – Podpora výuky evolučního návrhu využívajícího development.

Obsah

Kapitola 1

Úvod

1.1 Úvod

Se současným prudkým rozvojem elektrotechnického průmyslu dochází k nárůstu složitosti elektronických zařízení, což s sebou nese neustálé zvyšování stupně integrace při výrobě elektronických obvodů [53]. Vzhledem k fyzikálním omezením současných výrobních technologií nelze při sériové výrobě integrovaných obvodů (IO) zajistit bezchybnou produkci všech IO. Důležitým krokem výrobního procesu se tak stává posouzení, zda je vyrobený IO schopen správné funkce (zda jeho funkce odpovídá navržené implementaci). Včasnou detekcí a vyřazením defektních IO lze ušetřit značné ekonomické prostředky, které by bylo později nutné vynaložit na odhalení a odstranění závady v důsledku použití nekorektně fungujícího IO jako komponenty složitějšího systému. Vzhledem ke komplexnosti současných číslicových obvodů nelze většinou vystačit s jednoduchými testy aplikovanými přes primární vývody (vývody vyvedené z pouzdra IO). Jedním z cílů návrhářů je proto dosažení dobré testovatelnosti návrhu. Tento pojem je většinou chápán jako stav, umožňující bezproblémovou aplikaci diagnostických metod na obvod a dosažení vysokého pokrytí chyb. K zajištění dobré testovatelnosti je většinou nutné rozšíření návrhu o další pomocnou logiku, která umožní provádět test číslicového obvodu. Důležitým a velmi sledovaným parametrem se tak stává „cena testu“ a je pochopitelné, že snahou návrhářů je její minimalizace. Promyšleným návrhem je možné „cenu testu“ minimalizovat. Standardem pro návrh moderních číslicových obvodů se proto stává návrh s ohledem na snadnou testovatelnost (DfT) , kdy se o testování obvodu začíná uvažovat již v rané etapě jeho návrhu.

Dalším často sledovaným parametrem je „délka testu“. Časově kratší aplikace testu znamená možnost otestovat (a tím i vyrobit) více obvodů za stejnou časovou jednotku. Snahou návrhářů proto bývá dosáhnout co největší souběžnosti dílčích testů, čímž je možné docílit výrazného zrychlení aplikace testu oproti sekvenčnímu testu. Takový přístup však vyžaduje propracované metody plánování testu. U moderních obvodů se dnes také dostává do popředí kriterium nízkého příkonu. Obecně lze totiž během aplikace testů navržených pomocí konvenčních metod vysledovat vyšší příkon ve srovnání s běžným funkčním režimem činnosti obvodu [43]. Příkon je důležitým parametrem nejen u systémů s omezenými možnostmi napájení (např. systémy napájené z baterií), ale i u VLSI (Very Large Scale Integrated) systémů pracujících s vysokým hodinovým kmitočtem. Vyšší příkon má za následek vyšší tepelné ztráty a tudíž klade i vyšší nároky na systém chlazení. Dále se mohou projevovat negativní vlivy v důsledku tzv. „hot electron effect“ a tunelování. Vysoká hodnota procházejícího proudu může přetěžovat napájecí zdroj, způsobit značné úbytky napětí na napájecích linkách, způsobit přerušení spojů v důsledku jevu zvaného elektromigrace a

taktéž může ovlivnit funkci obvodu díky elektromagnetické indukci. Tyto jevy mohou při nedostatečně dimenzovaném návrhu obvodu negativně ovlivnit výsledky jeho testu. Dimenzování obvodu na maximální možný příkon během aplikace testu se jeví jako neefektivní, protože číslicový obvod bude po většinu svého životního cyklu provozován v běžném funkčním režimu činnosti a jeho parametry tak budou zbytečně předimenzovány. Z těchto důvodů jsou využívány metody pro redukci příkonu během aplikace testu. Principiálně je možné pro redukci příkonu použít i všeobecně rozšířené metody pro návrh nízkopříkonových obvodů. V takovém případě lze dosáhnout snížení absolutních hodnot příkonu, avšak disproporce mezi funkčním a diagnostickým režimem činnosti obvodu zůstává zachována. Tyto důvody vedly ke vzniku specializovaných optimalizačních metod, které umožňují dosáhnout redukce příkonu během aplikace testu. V této práci budou popsány nejpoužívanější z těchto metod. Obecně se redukce příkonu během aplikace testu dosahuje prodloužením testu, takže nedochází k poklesu energie odebrané ze zdroje. Některé z dále popsaných metod však umožňují redukovat i energii odebranou ze zdroje, takže mohou být využity např. pro snížení nákladů na energii během hromadného testování obvodů při výrobě.

Hlavním cílem této práce je návrh metodologie pro optimalizaci pořadí aplikace testovacích vektorů a zapojení registrů do řetězce scan umožňující dosáhnout redukce příkonu i odebrané energie během procesu aplikace testu. Snahou je dosáhnout kompatibility s moderními návrhovými systémy a možnost praktického využití této metodologie během standardního návrhového procesu. Dalšími požadavky je robustnost metodiky, možnost nasazení na komplexní VLSI obvody, návaznost na konkrétní technologickou knihovnu umožňující dosáhnout dostatečné přesnosti při ohodnocování kvality jednotlivých řešení, možnost nasazení na kombinační i sekvenční obvody (s plným řetězcem scan) a formální definice metodiky. Předpokladem je, že tento přístup umožní dosáhnout kvalitnějších výsledků ve srovnání s existujícími metodami.

1.2 Struktura disertační práce

Práce je rozčleněna následovně: V kapitole 1 je uveden stručný úvod do problematiky. Kapitola 2 popisuje princip návrhu obvodu pro snadnou testovatelnost. Jsou definovány jednotlivé pojmy a popsány všeobecně používané postupy. Kapitola 3 se zabývá problematikou příkonu obvodu během aplikace testu. Je uveden fyzikální model CMOS hradla, jsou definovány pojmy a matematické rovnice umožňující vyčíslení dynamické i statické složky příkonu včetně nejčastěji používaných zjednodušujících příkonových metrik. Kapitola 4 popisuje současný stav řešené problematiky a existující metody. Jednotlivé metody jsou kategoricky rozčleněny a je stručně popsán jejich princip. Kapitola se věnuje problematice stanovení příkonu a problematice redukce příkonu a odebrané energie během aplikace testu. Jsou popsány metody pro redukci dynamické i statické složky příkonu, metody modifikující strukturu číslicového obvodu, metody využívající optimalizace testu i kombinované metody. V popisu nechybí metody zaměřené na kombinační obvody ani metody zaměřené na obvody obsahující řetězec scan. Jsou prezentovány metody zaměřené na externí test i metody zaměřené na interní test. Tyto kapitoly slouží pro uvedení do problematiky. Jádro disertační práce tvoří následující kapitoly, jež jsou příspěvkem autora do dané problematiky. Kapitola 5 shrnuje cíle disertační práce. Kapitola 6 popisuje návrh vlastní metodiky. V kapitole je definován použitý formální model a jednotlivé algoritmy, pomocí kterých je postupně vystavěna celá metodika. Pro názornou demonstraci problematiky kapitola obsahuje množství příkladů. Kapitola 7 popisuje vytvořenou implementaci. Kapitola 8 diskutuje dosažené experimentální výsledky. Je zde též uvedeno srovnání s existujícími metodami. Kapitola 9

obsahuje shrnutí práce, jsou rozebrány výhody a nevýhody popsané metodologie i vlastní implementace a možnosti dalšího rozšíření práce.

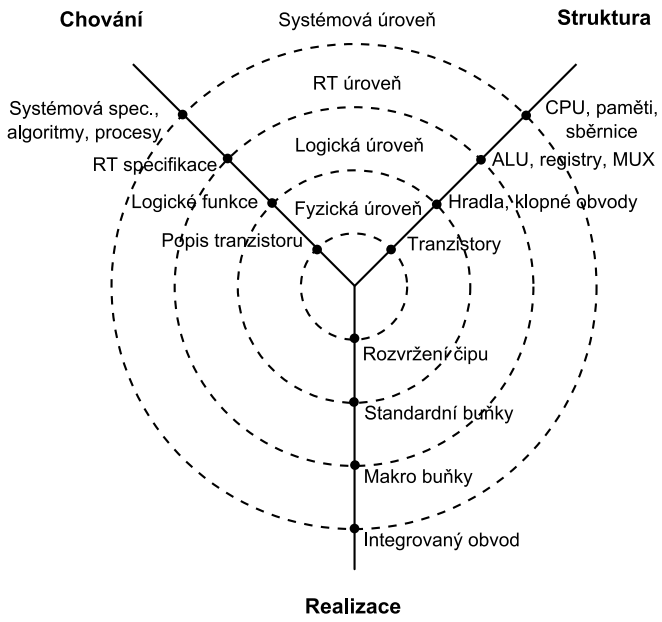
Kapitola 2

Návrh obvodů pro snadnou testovatelnost

2.1 Úrovně modelování číslicového obvodu

Vzhledem ke značné složitosti moderních číslicových obvodů je přímý návrh jejich struktury v dnešní době nemyslitelný, obzvlášť, když se v důsledku konkurenčního boje stává kritickým parametrem TTM (Time to Market – čas pro uvedení na trh). Při návrhu moderních číslicových obvodů je proto běžné použití vyšší úrovně abstrakce. Podle toho s jakou úrovní abstrakce pracujeme, lze rozlišit několik úrovní popisu. Nutno však podotknout, že počet a pojmenování těchto úrovní není často v literatuře jednotné. Zpravidla však lze identifikovat čtyři základní úrovně popisu a to *systémovou úroveň*, *úroveň meziregistrových přenosů* (RTL – Register Transfer Level), *úroveň hradel* a *fyzickou úroveň*. Nehledě na použitou úroveň abstrakce je možné zvolit různý způsob popisu. V tomto směru je možné rozlišit *popis chování*, *popis struktury* a *popis realizace*. Přehledně to lze celé ilustrovat např. pomocí tzv. Y-diagramu původně zavedeného Gajskim [17]. Y-diagram je zakreslen na obr. 2.1. V tomto diagramu jsou pomocí soustředných kružnic vyznačeny jednotlivé úrovně abstrakce, přičemž úroveň abstrakce vzrůstá od středu grafu směrem ven z grafu. Graf je pomocí úseček rozdělen na tři části, které reprezentují jednotlivé způsoby popisu.

Systémová úroveň je nejvyšší úrovní abstrakce. Návrhář tu definuje „čeho chce dosáhnout“, ale jednotlivé implementační detaily se hlouběji neřeší. Je třeba zmínit, že zobrazení mezi jednotlivými úrovněmi návrhu není injektivní – definovaná vyšší úroveň popisu může odpovídat mnoha různým implementacím v nižší úrovni popisu. V tomto smyslu lze lokálně označovat vždy příslušející vyšší úroveň abstrakce *specifikací* a k ní odpovídající nižší úroveň abstrakce *implementací* [16]. Popis chování na systémové úrovni sestává z algoritmického popisu chování, který může být vyjádřen pomocí přirozeného jazyka, programovacího (počítačového) jazyka, pomocí formálních modelů, grafických schémat, případně pomocí libovolných jiných popisných prostředků. V praxi se nejčastěji používá některý z HDL (Hardware Description Language – jazyk pro popis hardware), tedy programovacího jazyku rozšířeného o prostředky specifické pro popis hardware. Na systémové úrovni se při popisu chování identifikují jednotlivé systémové procesy a jejich vzájemné interakce. V praxi nejrozšířenější je zápis pomocí jazyků Verilog a VHDL, které pro popis jednotlivých procesů poskytují vhodné prostředky. Při popisu struktury se rozlišují základní stavební prvky obvodu. Zpravidla se jedná o dekompozici obvodu na prvky jako jsou procesory, paměti, sběrnice, atp. Pro většinu návrhářů je v tomto případě nejpřirozenějším prostředkem pro



Obrázek 2.1: Y graf

zápis grafické schéma, nicméně z hlediska dalšího zpracování nebývá nejvhodnějším. Jako vhodnější se jeví pro zápis struktury použít některého z jazyků HDL. V současné době existují i pokročilé návrhové systémy umožňující vizualizovat existující HDL kód nebo dokonce i z grafické reprezentace přímo generovat HDL kód, což celý proces zpřehledňuje. Z pohledu popisu realizace je důležitý počet a fyzické rozmístění vývodů na pouzdře obvodu, rozměry pouzdra, definované provozní podmínky (teplota, tlak, maximální povolené zrychlení, napájecí napětí, atp.).

Pokud uvažujeme systém multiplexovaných datových cest, tak systém na úrovni meziregistrových přenosů lze typicky rozdělit na dvě části - na část datových cest a obvodový řadič ovládajícího tok dat přes tyto cesty [41]. Při popisu chování se obvod na této úrovni typicky modeluje pomocí diagramu datového toku, ve kterém se využívají odpovídající matematické operace (násobení, dělení, sčítání, atp.). Při strukturálním popisu se používá prvků jako aritmeticko-logická jednotka, sčítačka, násobička, multiplexor, které jsou vzájemně odděleny pomocí registrů. Přestože se stále ještě jedná o poměrně vysokou úroveň abstrakce, je na této úrovni již k dispozici dostatek informací o struktuře vznikajícího obvodu, takže je možné se začít efektivně zabývat analýzou nejrůznějších vlastností obvodu, včetně úvah o diagnostice. Z pohledu popisu realizace začíná být důležitá technologická realizace jednotlivých prvků. Při tomto pohledu lze jednotlivé standardní prvky mapovat na technologickou knihovnu, kdy na každý prvek lze nahlížet jako tzv. „makro buňku“. Na tomto místě je nutné poznamenat, že ohledně pojmu úroveň meziregistrových přenosů nebývá vždy literatura jednotná. Nezřídka např. dochází k neshodám, co je ještě úroveň meziregistrových přenosů a co je již úroveň hradel.

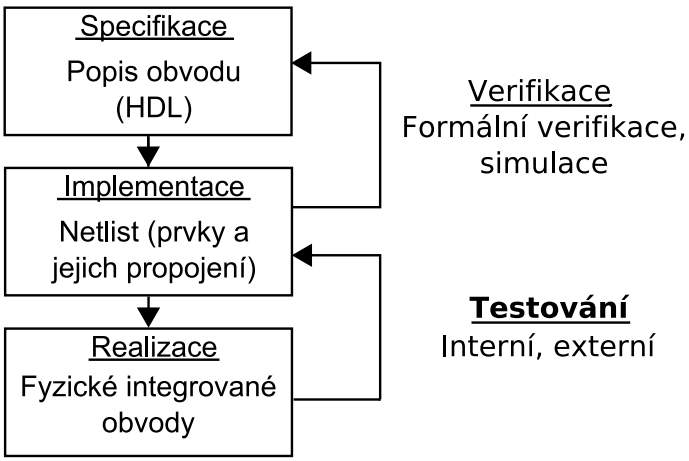
Pro popis chování obvodu na úrovni hradel se nejčastěji používá Booleovy algebry. Jednotlivé logické stavy se modelují pomocí hodnot „0“ a „1“. Chování obvodu se pak popisuje pomocí logických rovnic využívajících základních logických operátorů *and*, *or*, *not*, případně dalších. Při popisu struktury se používá dekompozice obvodu na jednoduché logické prvky realizující jednotlivé logické funkce. Při popisu realizace nás zajímá zejména

použitá technologie (např. CMOS) a reálné fyzikální vlastnosti jednotlivých prvků. Při tomto pohledu lze na každý prvek nahlížet jako na tzv. „buňku“.

Fyzická úroveň je nejnižší úrovní abstrakce. Je představována elementárními elektrotechnickými prvky (tranzistory, rezistory, kondenzátory). Chování obvodu na této úrovni lze popsat pomocí diferenciálních rovnic případně jejich zjednodušenou formou. Tento způsob je však velice komplikovaný a časově náročný. V praxi se používá jen velmi zřídka, jsou-li k tomu vážné důvody, např. potřebujeme-li přesně vyšetřit chování některého prvku při změně vnějších podmínek (např. teplota). Strukturální popis obvodu na této úrovni definuje propojení jednotlivých elementárních prvků. Při popisu realizace nás zajímá, jak bude prvek realizován „na křemíku“ (tzv. layout), kolik plochy čipu zabere, atp.

2.2 Etapy návrhu VLSI obvodů

Typický postup při návrhu VLSI obvodů je zachycen na obr. 2.2. Pod pojmem VLSI obvody jsou myšleny obvody obsahující více než 10^5 tranzistorů [18]. V tomto postupu lze rozlišit tři hlavní etapy návrhu - specifikaci, implementaci a realizaci (fyzickou výrobu čipu) [40; 70]. V první etapě je vytvořena systémová specifikace, ve které se definují požadavky na vlastnosti a funkci obvodu, strukturu rozhraní obvodu, pracovní kmitočet a další. Z této prvotní specifikace pak obvykle vychází specifikace jednotlivých podsystémů, kdy je původní rámcová specifikace hierarchicky rozšiřována a postupně zjemňována. Pokud máme již některé z podsystémů k dispozici (např. v nějaké knihovně prvků), nemusíme se jejich návrhem opětovně zabývat, čímž je možné některé návrhové kroky vynechat a tím výrazně uspořit návrhový čas. Takový postup se označuje termínem znovupoužitelnost návrhu a je často používaný při návrhu moderních obvodů. Pro specifikaci se v praxi nejčastěji používá některého z jazyků HDL. Jejich nespornou výhodou je, že umožňují v jednom návrhu současně pracovat s popisem struktury i chování, podle toho, co je v danou chvíli pro návrháře výhodnější.



Obrázek 2.2: Typické etapy při návrhu číslicových obvodů

V implementační etapě se přechází na nižší úroveň abstrakce - v tomto bodě většinou dochází k opuštění popisu chování obvodu a návrh je realizován na strukturální úrovni. Jsou identifikovány jednotlivé bloky implementující požadované funkce, je definováno propojení jednotlivých bloků a z těchto údajů se generuje seznam spojů (tzv. netlist) popisující

strukturu obvodu. Tento proces se nazývá syntéza a bývá většinou plně automatizován. V praxi se používá mapování na definovanou technologickou knihovnu prvků, k nimž už jsou přímo definovány odpovídající rozvržení pro jednotlivé realizační technologie. Nyní dochází k plánování čipu, kdy je vytvořeno výsledné rozvržení čipu. Na konci tohoto procesu jsou k dispozici veškeré podklady nutné pro zahájení výroby čipu.

Poslední etapou je pak už vlastní fyzická realizace, výsledkem této etapy jsou reálné IO. Při přechodech mezi jednotlivými návrhovými etapami je nutno ověřovat, zda nedošlo k zanesení nějaké chyby do procesu návrhu. Verifikací se zjišťuje, zda implementace odpovídá specifikaci. Implementaci lze verifikovat formálně nebo pomocí simulace. Často se používá několikastupňová verifikace, kdy se simuluje chování obvodu na různých úrovních abstrakce. Obdobným způsobem je nutné verifikovat realizaci oproti implementaci, k čemuž slouží testování. Testováním se tedy ověřuje, zda fyzická realizace skutečně odpovídá implementaci [13]. Tato práce je orientována na proces testování IO. Předpokládá se, že byla realizována korektní implementace.

2.3 Základní pojmy technické diagnostiky

V této práci je využívána terminologie z [27]. Termín diagnostika byl původně převzat z lékařství, protože cíle i metody používané v technické diagnostice (dále jen diagnostice) jsou dosti podobné. Úlohou diagnostiky je zjistit, zda je zkoumaný objekt schopen korektní funkce. Pokud není, metody diagnostiky umožňují nalézt příčinu chybné funkce. Hojně bývá využíváno deduktivních metod, protože zpravidla není možné přímo zjistit stav zkoumaného objektu (přímo ověřit funkci všech jeho součástí). Využívá se proto odvozování ze souvislostí mezi pozorovanými změnami chování objektu a předpokládanými příčinami tohoto chování.

Z hlediska diagnostiky je taktéž důležitá granularita pohledu. Pojmem testovaná jednotka bývá proto označována právě zkoumaná část systému. Testovaná jednotka má definované rozhraní, přes které může komunikovat s okolím. Toto rozhraní sestává ze vstupů a výstupů, které bývají označovány jako primární. Jednou z úloh diagnostiky je u testované jednotky rozlišit poruchový a bezporuchový stav. Poruchový stav je u technických zařízení nežádoucí a bývá zapříčiněn alespoň jednou poruchou. V poruchovém stavu není zajištěna korektní funkce testované jednotky (ale není ani úplně vyloučena např. pro jistou podmnožinu funkci neovlivněných poruchou). Dalšími úlohami diagnostiky může být určení počtu poruch, typu poruch, případně přesná lokalizace poruch. Popsané úlohy diagnostiky se řeší pomocí diagnostických testů. Návrh a aplikace diagnostických testů bývá označována souhrnným pojmem testování.

2.4 Testování číslicových systémů

Testování bývá uplatňováno během celého životního cyklu číslicového systému. Typický číslicový systém je zpravidla možné rozdělit na jednotlivé moduly, jež jej prostorově i logicky dekomponují na samostatné menší celky ([27]). V takovém případě diagnostické testy nejčastěji slouží k identifikaci poruchových modulů. Oprava systému potom spočívá ve výměně poruchových modulů. Z tohoto pohledu lze rozlišit tzv. periodické testování (též offline testování), která bývá prováděno v pravidelných intervalech za účelem udržení systému v bezporuchovém stavu ([27]). Toto vyžaduje odstavení systému během aplikace testu a v této době systém nemůže vykonávat běžnou činnost. Proto bývají tyto testy nejčastěji prováděny v přestávkách mezi výpočty a pochopitelně snahou návrhářů je jejich délku mi-

nimalizovat. Tento přístup je často volen vzhledem k příznivým ekonomickým nákladům. Jistou nevýhodou však je riziko chybného výpočtu v době mezi periodickými testy, než dojde k odhalení poruchy. s Uvedený nedostatek odstraňuje tzv. průběžné testování (též online testování), kdy dochází k samočinnému testování systému během jeho běžné funkce ([27]). K tomu se často využívá tzv. hlídacího obvodu, který průběžně kontroluje správnou činnost systému. Kvalita a složitost hlídacího obvodu jsou u těchto aplikací kritickými parametry. Z ekonomických důvodů (a i z důvodů spolehlivosti) bývá cílem dosáhnout nižší složitosti hlídače ve srovnání s hlídaným systémem. Jako další možnosti se využívá replikace komponent systému, kdy redundantní komponenty systému vykonávají shodnou funkci. Výsledky od jednotlivých komponent jsou porovnávány a správný výsledek je vybrán dle majority shodných výsledků. Takto konstruované systémy jsou sice velmi odolné vůči poruchám, nicméně jsou značně nákladné, proto tento přístup bývá využíván jen v opodstatněných kritických aplikacích z oblasti lékařství, kosmonautiky, armády, apod.

2.5 Testování číslicových obvodů

Obdobně jako u číslicových systému, které je možné dekomponovat na jednotlivé moduly, i tyto moduly lze dále dekomponovat na jednotlivé diskrétní součástky, které mohou být také poruchové a je tedy třeba je otestovat minimálně před osazením na modul. Z používaných diskrétních součástek jsou nejkomplexnější a tedy i nejnáročnější z hlediska testování integrované obvody. Dnešní technologie umožňují výrobu i celých systémů na jednom čipu (SoC – Systems On Chip). Takové systémy na čipu jsou typicky IO vytvořené technologií s velmi vysokým stupněm integrace (VLSI), které jsou složeny z miliónů tranzistorů a obsahují širokou škálu různorodých modulů (např. procesor Dual Core Itanium 2 obsahuje 1,7 miliardy tranzistorů [62]).

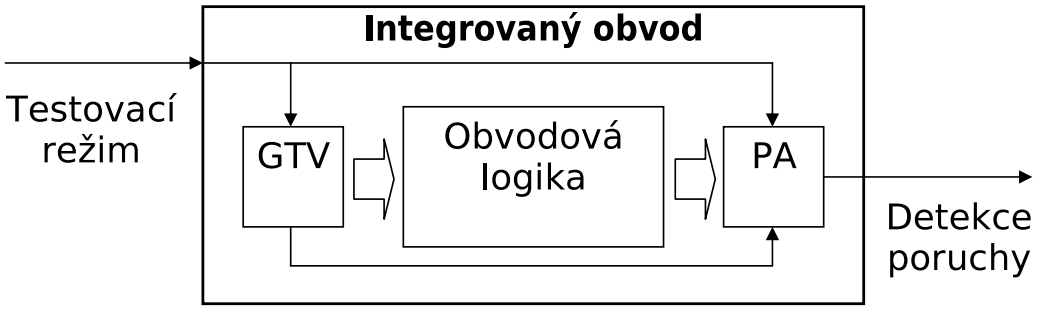
Testování umožňuje identifikovat poruchové IO. V praxi se testování používá k odhalení poruch, ke kterým mohlo dojít při výrobě obvodu, ale může být použito i pro ověření, zda nedošlo k poškození obvodu během jeho životního cyklu. Při výrobě obvodu se mohou vyskytnout různé poruchy, jejich příčiny a četnost výskytu závisí na použité výrobní technologii. Časté jsou poruchy způsobené nekvalitními spoji, nečistotami materiálu, přemostěním (zkraty), aj. Čím dříve dojde k odhalení těchto poruch, tím nižší jsou výsledné náklady na jejich odstranění. Snahou návrhářů je proto testem zachytit co nejvíce takovýchto poruch.

Testování lze dále rozdělit na parametrické nebo logické. Parametrické testování odhaluje poruchy vyhodnocováním měřitelných fyzikálních parametrů během funkce obvodu. Např. je sledován odběr proudu ze zdroje [9] nebo teplota v různých částech obvodu [5]. Logické testování modeluje poruchy na logické úrovni abstrakce. Nejpoužívanějšími poruchovými modely jsou trvalá „0“, trvalá „1“, přemostění, rozpojený spoj a dynamické poruchy zpoždění. Podle způsobu aplikace testu na obvod lze rozlišit vnější (externí) testování s využitím samočinného vnějšího testeru (ATE – Automatic Test Equipment) a vnitřní (interní nebo též vestavěné) testování (BIST – Built-In Self-Test), případně jejich kombinace.

2.5.1 Vnitřní testování

Vnitřní testování nevyžaduje žádné vnější zařízení kromě napájení a zdroje synchronizačního (hodinového) signálu. Veškerá logika nutná pro generování i aplikaci testu je již přímou součástí navrženého obvodu. To sice zvyšuje výrobní náklady, ale zase odpadá nutnost investice do nákladného testeru. Výhodou je, že takovýto obvod může pak snadno testovat

i zákazník mimo výrobní linku, případně lze takovýto obvod testovat i po vestavění do cílového zařízení např. při startu systému (POST – Power On Self Test). Při tomto typu testování jsou vstupní data testu (testovací vektory) generovány uvnitř obvodu pomocí generátoru testovacích vektorů (GTV, anglická zkratka TPG – Test Pattern Generator). Při generování testovacích vektorů se používají různé strategie – pseudonáhodná, úplný výčet možností, neúplný výčet možností nebo kombinace předcházejících přístupů. Odezva obvodu je vyhodnocována pomocí příznakového analyzátoru (PA, anglická zkratka SA – Signature Analyzer). Princip lze vidět na obr. 2.3. Pro konstrukci GTV a PA se nejčastěji používá lineární posuvný registr se zpětnou vazbou (LPRZV, anglická zkratka LFSR – Linear Feedback Shift Register). Chování těchto obvodů je dnes již matematicky dobře popsáno [64]. Při konstrukci GTV a PA je také možné využít celulárních automatů [9].

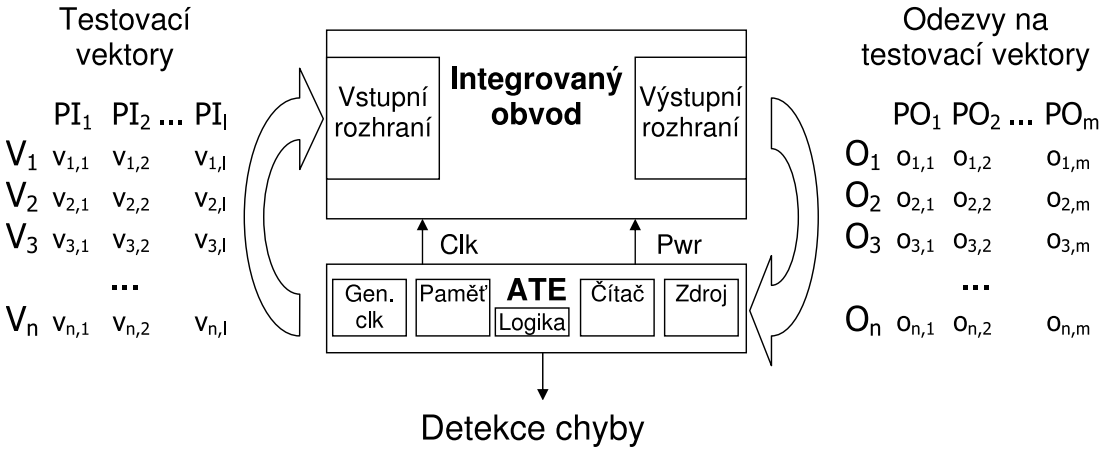


Obrázek 2.3: Princip vnitřního testování

2.5.2 Vnější testování

Při externím testování se využívá samočinného testeru (ATE), který se připojuje na primární vstupy/výstupy (vývody) testovaného obvodu. Provedení ATE a zejména provedení konektorů musí být dostatečně robustní, protože se předpokládá připojování a odpojování značného množství obvodů (řádově miliony) během celého životního cyklu ATE. Pořizovací náklady ATE jsou značné, jedná se totiž o velmi komplikovaná zařízení. Běžné ATE obsahuje centrální procesor, který řídí průběh celého testu dle určitého programu, zdroje potřebných napájecích napětí, paměť testovacích vektorů, paměť referenčních odezev, přesnou časovou základnu a moduly pro měření fyzikálních veličin. Současné ATE jsou schopny měřit napětí s přesností na mV a čas s přesností na stovky pikosekund [9]. Tento způsob testování se vzhledem k nákladům na ATE používá výhradně při výrobě IO. Na obr. 2.4 je schématicky znázorněno použití ATE. Na primární vstupy PI_1-PI_l testovaného IO ATE postupně přivádí jednotlivé testovací vektory V_1-V_n. Odezvy na testovací vektory O_1-O_n jsou snímány na primárních výstupech IO PO_1-PO_m. Hodnoty $v_{i,j}$ $(i = 1 \ldots n,\ j = 1 \ldots l)$ jsou logické hodnoty („0“/„1“) jednotlivých bitů testovacích vektorů. Hodnoty $o_{i,j}$ $(i = 1 \ldots n,\ j = 1 \ldots m)$ jsou logické hodnoty („0“/„1“) jednotlivých bitů odezev. Jednotlivé odezvy IO jsou porovnávány s referenčními vzorky z paměti ATE. V případě, že některá odezva nesouhlasí s referenčním vzorkem, je detekována porucha. Výhodou ATE je, že kromě logického testu může současně provádět i parametrický test, např. měření teploty IO a měření odebíraného proudu pro jednotlivé aplikované testovací vektory. Obdobným způsobem může ATE sledovat i dynamické parametry jako zpoždění, zkreslení logického signálu, atp. Stačí jen rozšířit ATE o patřičnou měřicí jednotku, do paměti ATE přidat patřičné položky a

upravit řídicí program. Vzhledem ke složitosti současných IO a omezenému počtu primárních vstupů/výstupů, efektivní nasazení ATE vyžaduje provedení návrhu obvodu s ohledem na snadnou testovatelnost (DfT). technik při návrhu obvodu. Tato práce se dále orientuje na externí test obvodů označovaných jako ASIC (Application Specific Integrated Circuits – aplikačně specifické integrované obvody).



Obrázek 2.4: Vnější testování

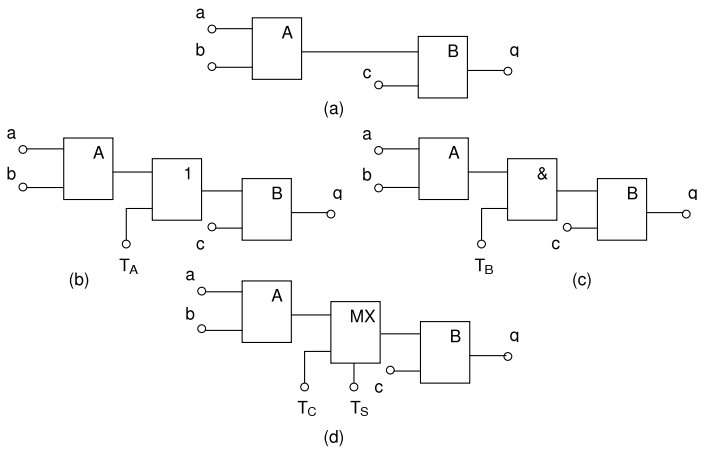
2.5.3 Plánování testu

Plánování testu je proces, jehož cílem je nalezení takového způsobu aplikace testu, který umožní efektivní využití všech dostupných zdrojů během aplikace testu. Plánování testu se uplatňuje zejména tehdy, pokud máme k dispozici omezený počet GTV, PA, sběrnic pro přenos diagnostických dat nebo při použití částečného řetězce scan, kdy je nutné pro přenos diagnostické informace využívat i jiných mechanismů [59] např. tzv. mechanismu I-cest [4]. Ve většině případů je cílem naplánovat test takovým způsobem, aby bylo možné otestovat všechny požadované prvky pokud možno v nejkratším možném časovém intervalu. Dále je možné při plánování testu uplatnit i další optimalizace dle nejrůznějších parametrů. Jednou z dalších možností je např. optimalizace plánu testu pro nízký příkon.

2.6 Metody zlepšení testovatelnosti

Mezi klasické metody zlepšení testovatelnosti patří metody vkládání testovacích bodů. Tyto metody lze použít v případech, kdy je třeba vložit nebo sejmout logickou hodnotu na/z některého vnitřního uzlu obvodu, který není řiditelný/pozorovatelný z primárních vstupů/výstupů obvodu. Jedná se o tzv. metody ad-hoc, protože jsou zpravidla aplikovány až po skončení etapy návrhu. Lze rozlišit dva typy testovacích bodů – řídicí a pozorovací. Pozorovací body lze implementovat pomocí vodiče přímo vyvedeného na primární výstup číslicového obvodu. Implementace řídicích bodů vyžaduje rozšíření obvodu o další logické prvky. Na obr. 2.5(a) je zobrazena část logického obvodu, kde je třeba mezi prvky A a B vložit řídicí bod. Lze identifikovat tři typy řídicích bodů: body umožňující nastavení na hodnotu logická „1“ – implementace pomocí hradla logického součtu (obr. 2.5(b), hodnota se nastaví přivedením „1“ na vstup T_A), body umožňující nastavení na hodnotu logická „0“ –

implementace pomocí hradla logického součinu (obr. 2.5(c), hodnota se nastaví přivedením „0“ na vstup T_B) a plně řiditelné testovací body – implementace pomocí multiplexoru (obr. 2.5(d), hodnota se přivede na vstup T_C a pomocí vstupu T_S se nastaví testovací režim).

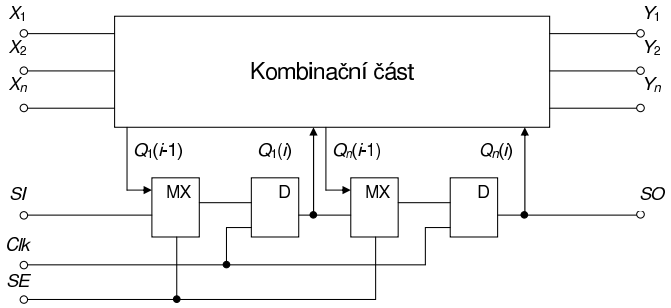


Obrázek 2.5: (a) číslicový obvod a jeho modifikace umožňující (b) vkládání hodnoty logická „1“, (c) logická „0“, (d) úplné řízení

Nedostatkem uvedené metody je, že pro každý řídicí/pozorovací bod vyžaduje samostatný primární vstup/výstup. V případě, že není vyžadováno testování na rychlosti čipu (tzv. at-speed testing), lze využít multiplexorů/demultiplexorů pro redukci počtu primárních vstupů/výstupů využitých pro testovací body. V tomto případě je možné pomocí n_1+1 primárních vývodů řídit či pozorovat $n_2 = 2^{n_1}$ testovacích bodů. Přestože dnes již existuje sofistikovanější metody pro zlepšení testovatelnosti, metody vkládání testovacích bodů jsou stále využívány v některých specifických aplikacích.

Modernějšími metodami jsou metody využívající strukturovaného návrhu . Principem těchto metod je rozdělení číslicového obvodu na kombinační a sekvenční část. Sekvenční část je následně modifikována tak, aby bylo možné řídit/pozorovat vnitřní stavy jednotlivých prvků. Díky možnosti nastavení vnitřních stavů, lze pak celý obvod testovat jako kombinační. Princip budeme ilustrovat na obvodu využívajícím jako paměťové prvky klopné obvody typu D. Každý klopný obvod rozšíříme o multiplexor (viz obr. 2.6). Na adresní vodiče multiplexorů přivedeme signál SE rozlišující běžný funkční režim činnosti obvodu ($SE = 0$) od testovacího režimu ($SE = 1$). V běžném funkčním režimu budou klopné obvody sloužit jako paměť vnitřního stavu. V testovacím režimu do nich budou přes dodatečné testovací vstupy sériově načítány a přes dodatečné testovací výstupy sériově vyčítány vnitřní hodnoty. Pro úsporu primárních vývodů lze tyto pomocné testovací vstupy/výstupy zapojit do posuvného registru. Tento posuvný registr bývá označován termínem sériový řetězec scan (existují však i paralelní varianty využívající např. paměti RAM, tato řešení však zpravidla vyžadují větší plochu čipu a nejsou tak rozšířená). Díky tomu lze vystačit se třemi přídavnými primárními vývody (SI, SO, SE), viz obr. 2.6.

Na obr. 2.6 X_1, X_2, …, X_n jsou primární vstupu, Y_1, Y_2, …, Y_n jsou primární výstupy, $Q_1(i-1)$, …, $Q_n(i-1)$ jsou vnitřní stavy obvodu v čase $i-1$, $Q_1(i)$, …, $Q_n(i)$ jsou vnitřní stavy obvodu v čase i, SI je sériový vstup řetězce scan, SO je sériový výstup řetězce scan, SE je vstup sloužící k volbě mezi normálním funkčním režimem a testovacím režimem, Clk je vstup synchronizačního signálu. Výhodou strukturovaných metod návrhu



Obrázek 2.6: Princip metod strukturovaného návrhu

je možnost jejich jednoduchého nasazení již ve fázi návrhu obvodu. Nevýhodou je nárůst plochy čipu o pomocné logické prvky a prodloužení doby aplikace testu (pro n prvkový řetězec scan je třeba pro každé načtení/sejmutí vnitřních stavů $n + 1$ synchronizačních pulsů, n synchronizačních pulzů pro sériový posuv hodnot řetězcem scan a jeden synchronizační pulz pro paralelní načtení odezvy obvodu do řetězce scan). Nárůst plochy čipu je možné omezit použitím částečného řetězce scan. V takovém případě však není oddělena kombinační část obvodu od sekvenční a řetězce scan je použito jen pro snížení sekvenční hloubky testu. Generování i aplikace testu pro takové obvody většinou mnohem náročnější. Částečného řetězce scan je využito např. u metod publikovaných v [23; 32; 47; 55]. Existují dokonce i komerční nástroje umožňující vkládání částečného řetězce scan do návrhu obvodu a generování testovacích vektorů pro částečný scan, toto umožňují např. nástroje od firmy Mentor Graphics [39]. Dobu aplikace testu lze omezit rozdělením řetězce scan na více menších řetězců, samozřejmě za cenu složitějšího řízení testu.

Mezi nejrozšířenější metody strukturovaného návrhu číslicových obvodů patří metoda LSSD (Level Sensitive Scan Design) [73]. U této metody jsou scan registry sestaveny z klopných obvodů reagujících na úroveň (klopné obvody typu latch). Zápis do klopného obvodu probíhá pomocí dvou časově disjunktních synchronizačních pulsů.

2.7 Analýza testovatelnosti

Cílem metod analýzy testovatelnosti je kvantitativní číselné ohodnocení diagnostických vlastností obvodu. Z tohoto pohledu je testovatelnost vnímána jako charakteristika zohledňující náklady spojené s testováním číslicového obvodu. V používané terminologii a definicích panovala dlouhou dobu nejednotnost. Sjednocení zavádí až norma IEEE P1522. Většina metod analýzy testovatelnosti je založena na výpočtu koeficientů vyjadřujících řiditelnost a pozorovatelnost jednotlivých obvodových uzlů či prvků. Dle hodnot koeficientů řiditelnosti a pozorovatelnosti je pak vyčíslena testovatelnost. Přesný postup vyčíslení řiditelnosti, pozorovatelnosti a testovatelnosti závisí na konkrétní použité metodě.

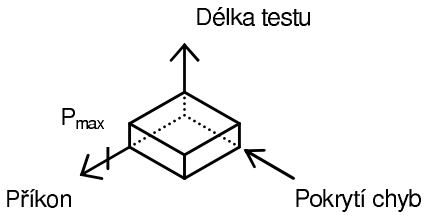
Metody analýzy testovatelnosti lze rozdělit dle úrovně návrhu, se kterou pracují. Na úrovni hradel je často používanou metodou metoda SCOAP (Sandia Controllability Observability Analysis Program) [24]. Míry řiditelnosti a pozorovatelnosti u ní reflektují obtížnost řízení resp. pozorování konkrétních jednobitových hodnot na vnitřních uzlech obvodu, přičemž vyšší hodnoty řiditelnosti resp. pozorovatelnosti indikují vyšší obtížnost této činnosti. Na úrovni meziregistrových přenosů je známá metoda CoPS (Cost-based Partial Scan) [46; 47]. Základem metody je výpočet nákladů spojených s aplikací testu na obvod s

částečným řetězcem scan, který je popsaný na úrovni hradel. Pro každý vodič v obvodu je metodou spočítaná tzv. cena detekovatelnosti poruchy, zohledňující cenu řiditelnosti, sekvenční hloubku a cenu pozorovatelnosti. Mezi další metody lze zařadit metodu definovanou v [33; 34], která využívá analýzy I-cest a klasifikace registrů na TIR (Test Input Register), TOR (Test Output Register), TDR (Test Driver Register), TRV (Test Receiver Register) a její obecnější rozšíření publikované v [63]. Na vyšší úroveň popisu lze zařadit např. systém SATAN (Systems Automatic Testability Analysis) [45], který byl vytvořen pro analýzu testovatelnosti a funkční specifikaci testů. Systém SATAN využívá grafového modelu přenosu informací obvodem. Uzly grafu reprezentují vývody a moduly obvodu, hrany grafu reprezentují různé režimy přenosu informací mezi uzly. Tok informací modelem je realizován po informačních cestách ze vstupu obvodu na jeho výstupy, přičemž během toku jsou informace transformovány moduly vyskytujícími se na těchto cestách. Testovatelnost obvodu je pak stanovena na základě odhadu hodnot řiditelnosti a pozorovatelnosti komponent v různých tocích.

Kapitola 3

Příkon obvodu

Při návrhu diagnostických testů je snahou návrhářů dosáhnout maximálního pokrytí chyb, minimální doby aplikace testu a nízkého podílu přídavné obvodové logiky. Bohužel všechny tyto cíle nelze splnit současně a je nutné hledat vhodná kompromisní řešení. Do popředí zájmu návrhářů se však dostává i další parametr – příkon obvodu. Vztah mezi příkonem obvodu, dobou aplikace testu a pokrytím chyb je možné ilustrovat např. pomocí obr. 3.1. Na tomto obrázku je naznačen příklad řešení a vztahy mezi jednotlivými parametry řešení. Každé řešení lze modelovat kvádrem, přičemž rozměry jeho stran definují hodnoty jednotlivých parametrů. Pro optimalizaci parametrů řešení je možné měnit rozměry stran kvádru, avšak objem kvádru musí zůstat konstantní. Tedy např. snížení příkonu lze dosáhnout prodloužením testu nebo aplikací testu s nižším pokrytím chyb. Hodnota P_{max} na obr. 3.1 představuje maximální povolený příkon (limit čipu), který nesmí být překročen. Obr. 3.1 slouží pouze pro ilustraci problematiky, měřítka os nejsou shodná a v praxi jsou navíc vztahy mezi jednotlivými parametry ovlivněny použitou implementační technologií, typem testu, strukturou testovaného obvodu a není tudíž možné jednotlivé parametry libovolně měnit. Zobrazení trojrozměrného prostoru řešení pro několik reálných obvodů lze nalézt např. v [42].

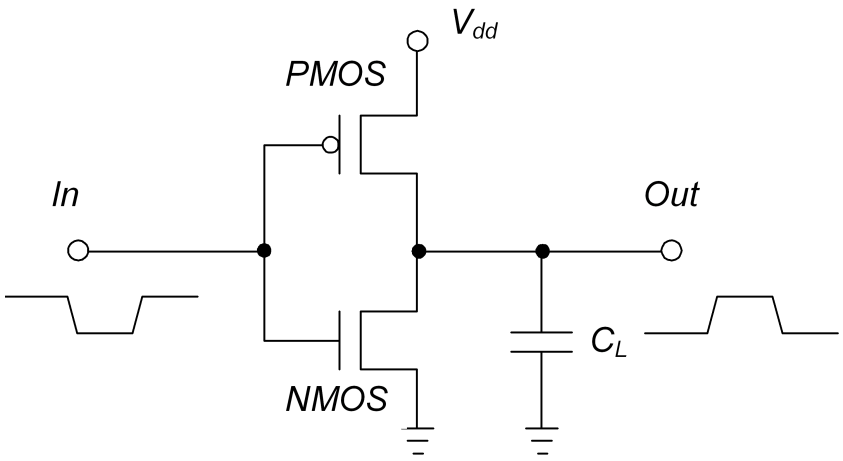


Obrázek 3.1: Ilustrace problému

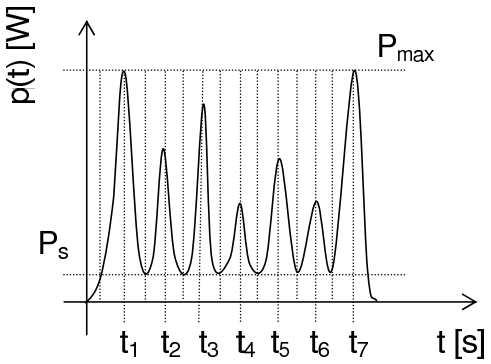
3.1 Příkon a odebraná energie

Při vyšetřování příkonu hradla je nutné se zaměřit na konkrétní realizační technologii. V této práci bude dále uvažována technologie CMOS jakožto v současné době dominantní technologie pro výrobu VLSI obvodů s více než 10^5 tranzistory (dle [43]). Schéma typického CMOS hradla lze vidět na obr. 3.2. Pokud vyneseme hodnotu okamžitého příkonu testovaného prvku do grafu v závislosti na čase, získáme spojitý průběh, jak lze vidět v grafu na

obr. 3.3. V tomto grafu je zobrazen průběh příkonu pro testovaný obvod během aplikace testovacích vektorů V_1 až V_7 v časech t_1 až t_7, $t_i = \frac{i}{f}$, kde $i = \langle 1; 7 \rangle$ je shodné s pořadovým číslem pulzu a f je frekvence synchronizačních pulzů. Zobrazený průběh odpovídá strategii „test per clock“ (TPC), kdy je při každém synchronizačním pulzu aplikován jeden testovací vektor a sejmuta odpovídající odezva. Podobné průběhy lze obdržet i v případě použití řetězce scan, kdy se jedná o testovací strategii „test per scan“ (TPS) [44], při které jednotlivé synchronizační pulzy řídí řetězec scan a krok testu sestává z několika synchronizačních pulzů.



Obrázek 3.2: Hradlo CMOS



Obrázek 3.3: Reálný průběh příkonu při aplikaci testovacích vektorů

Příkon obvodu lze potom zapsat pomocí rovnice (3.1) (převzato z [48], [53]).

$$p(t) = p_s(t) + p_d(t) \tag{3.1}$$

V rovnici (3.1) značí $p(t)$ příkon obvodu v čase t. $p(t)$ lze rozložit na statickou složku složku $p_s(t)$ a dynamickou složku $p_d(t)$.

3.1.1 Statická složka P_s

Složku $p_s(t)$ lze za daných provozních podmínek považovat za konstantní v čase, lze ji tedy značit P_s. P_s lze vyčíslit pomocí rovnice (3.2).

$$P_s = (I_{subtreshold} - I_{diode})U_{dd} \tag{3.2}$$

V rovnici (3.2) I_{diode} je zpětný diodový proud mezi difúzními oblastmi a substrátem. Tento proud bývá velice malý (typicky femto ampéry). Dále $I_{subthreshold}$ je ztrátový proud tranzistoru a lze jej určit z rovnice (3.3), U_{dd} je napájecí napětí.

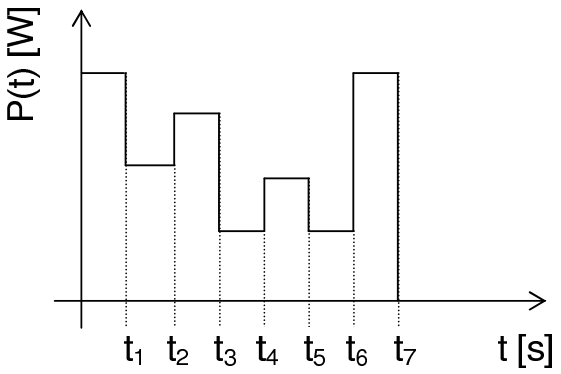
$$I_{subtreshold} = KW_{eff}e^{\frac{U_{in}-U_T}{S}} \tag{3.3}$$

V rovnici (3.3) K, S jsou konstanty závisející na použitých tranzistorech, W_{eff} je efektivní šířka kanálu tranzistoru. U_{in} je velikost vstupního napětí, U_T je velikost prahového napětí.

Složka P_s exponenciálně roste s klesající hodnotou U_T. U většiny v současnosti (rok 2008) nejpoužívanějších technologií složka P_s nepřesahuje 50 % z celkové hodnoty příkonu. U značného množství obvodů vyrobených staršími, nicméně stále velice rozšířenými technologiemi, tvoří dokonce méně než 10 % z celkové hodnoty příkonu. Složka P_s je pro danou obvodovou strukturu konstantní v čase bez závislosti na průběhu vstupních signálů. Z těchto důvodů jí v literatuře většinou není věnována příliš pozornost. Složka P_s je vyznačena i na příkladu na obr. 3.3.

3.1.2 Dynamická složka $p_d(t)$

Složka $p_d(t)$ závisí na stavu obvodu v čase t. U technologie CMOS dosahuje $p_d(t)$ největších hodnot během přechodu mezi jednotlivými logickými stavy ($0 \rightarrow 1$, $1 \rightarrow 0$). Omezíme-li se na střední hodnotu dynamické části příkonu P_d v intervalu $\left\langle \frac{(i-1)}{f}; \frac{i}{f} \right)$ [s], lze P_d u obvodu složeného z CMOS hradel zapsat jako součet složek P_{SW} a P_{SC} pomocí rovnice (3.4) (převzato z [48], [53]). Ze spojitého průběhu příkonu zobrazeného na obr. 3.3 potom dostáváme diskrétní průběh zobrazený na obr. 3.4.



Obrázek 3.4: Diskrétní průběh příkonu

$$P_d(i) = P_{SW}(i) + P_{SC}(i) \tag{3.4}$$

Složka P_{SW} bývá označována jako tzv. příkon přepínaných kapacit, který je způsoben nabíjením a vybíjením zátěžových kapacit. Lze ji vypočíst z rovnice (3.5). Složku P_{SC} tvoří tzv. příkon způsobený krátkým spojením, který je způsoben průchodem zkratového proudu $U_{dd} \rightarrow GND$ během krátké doby při přechodu CMOS hradel z jednoho logického stavu do druhého, kdy jsou otevřeny oba tranzistory NMOS i PMOS současně (viz obr. 3.2). P_{SC} lze vypočíst z rovnice (3.6).

$$P_{SW}(i) = \frac{1}{2} C_L U_d^2 n(i) f \tag{3.5}$$

V rovnici (3.5) je význam symbolů následující: C_L je kapacita hradel. U_{dd} je napájecí napětí, $n(i)$ je počet překlopení $(0 \rightarrow 1,\ 1 \rightarrow 0)$ v obvodu při přechodu ze stavu v čase $\frac{(i-1)}{f}$ do stavu v čase $\frac{i}{f}$, f je frekvence synchronizačního signálu.

$$P_{SC}(i) = K(U_{dd} - 2U_T)^3 \tau n(i) f \tag{3.6}$$

V rovnici (3.6) je význam symbolů následující: K je konstanta závisející na typu použitých tranzistorů, U_T je velikost prahového napětí, τ je délka trvání nástupných/sestupných hran signálu, U_{dd} je napájecí napětí, $n(i)$ je počet překlopení $(0 \rightarrow 1,\ 1 \rightarrow 0)$ v obvodu při přechodu ze stavu v čase $\frac{(i-1)}{f}$ do stavu v čase $\frac{i}{f}$, f je frekvence synchronizačního hodinového signálu. Dosazením 3.6, 3.5 do 3.4 a 3.4 do 3.1 dostáváme výsledný vzorec 3.7.

$$P(i) = P_s + n(i) f \left(\frac{1}{2} C_L U_{dd}^2 + K(U_{dd} - 2U_T)^3 \ \tau \right) \tag{3.7}$$

Tento model lze dále rozšířit a střední hodnotu příkonu během časového intervalu $\left\langle \frac{i_1}{f}; \frac{i_2}{f} \right)$ [s] $(i_1 < i_2)$ je možné zapsat pomocí rovnice (3.8).

$$P_{\langle i_1; i_2)} = \frac{\sum_{i=i_1+1}^{i_2} P(i)}{i_2 - i_1} \tag{3.8}$$

Energii odebranou během časového intervalu $\left\langle \frac{i_1}{f}; \frac{i_2}{f} \right\rangle$ [s] $(i_1 < i_2)$ lze obecně zapsat pomocí rovnice (3.9).

$$E_{\left\langle \frac{i_1}{f}; \frac{i_2}{f} \right\rangle} = \int_{\frac{i_1}{f}}^{\frac{i_2}{f}} p(t)\, dt \tag{3.9}$$

Zaměříme-li se na střední hodnotu příkonu, lze s využitím rovnice (3.7) energii odebranou ze zdroje během časového intervalu $\left\langle \frac{i_1}{f}; \frac{i_2}{f} \right)$ [s] $(i_1 < i_2)$ zapsat pomocí rovnice (3.10).

$$E_{\langle i_1; i_2)} = \frac{i_2 - i_1}{f} \sum_{i=i_1+1}^{i_2} P(i) \tag{3.10}$$

3.1.3 Zpoždění

Znalost zpoždění obvodových prvků je důležitá pro stanovení počtu překlopení v obvodu v důsledku hazardů a nalezení kritické cesty v obvodu. Pro výpočet zpoždění hradla t_d je možné použít rovnici (3.11).

$$t_d = \frac{\left(\frac{L_n}{K_n W_n} + \frac{L_p}{K_p W_p}\right) C_L \; U_{dd}}{(U_{dd} - U_T)^\alpha} \tag{3.11}$$

V rovnici (3.11) je α saturační zpoždění, které se pro technologii CMOS 0.18 μm pohybuje kolem hodnoty 1,3. L_n, W_n, L_p a W_p jsou délky a šířky MOS tranzistorů typu n a p, K_n a K_p jsou konstanty závisející na těchto tranzistorech. Ostatní symboly byly vysvětleny v předchozích rovnicích.

3.1.4 Zjednodušující metriky

Rovnice (3.7) je příliš komplikovaná pro výpočet, což není výhodné v případě složitých obvodů nebo v případě velkého množství vstupních dat. Z těchto důvodů se zejména pro účely relativního porovnání jednotlivých optimalizačních metod používají různé zjednodušující metriky, jež budou definovány v následujícím textu. Za předpokladu, že jsou modifikována pouze vstupní data a parametry obvodové realizace se nemění, lze metody porovnávat podle parametru NTC (Number of Transition Count), který lze určit z rovnice (3.12). Přesnější metriky jsou založeny na výpočtu WNTC (Weighted Number of Transition Count) rovnice (3.13) [43] a WSA (Weighted Switching Activity) rovnice (3.14) [15], které uvažují i kapacitu zátěže.

$$NTC = \sum_{i=1}^{N_C} n(i) \tag{3.12}$$

V rovnici (3.12) $n(i)$ značí počet překlopení $0 \rightarrow 1$, $1 \rightarrow 0$ při přechodu obvodu ze stavu v čase $\frac{i-1}{f}$ do stavu v čase $\frac{i}{f}$, N_C značí celkový počet synchronizačních pulsů během aplikace testu.

$$WNTC = \sum_{i=1}^{N_C} \sum_{j=1}^{N_G} n_j(i) F_j \tag{3.13}$$

V rovnici (3.13) $n_j(i)$ označuje počet překlopení v uzlu j při přechodu obvodu ze stavu v čase $\frac{i-1}{f}$ do stavu v čase $\frac{i}{f}$. F_j určuje počet rozvětvení uzlu j (tzv. fan out factor), N_C značí celkový počet synchronizačních hodinových pulsů během aplikace testu, N_G je celkový počet uzlů v obvodu.

$$WSA = \sum_{i=1}^{N_C} \sum_{j=1}^{N_G} n_j(i) C_j \tag{3.14}$$

V rovnici (3.14) $n_j(i)$ určuje počet překlopení v uzlu j při přechodu obvodu ze stavu v čase $\frac{i-1}{f}$ do stavu v čase $\frac{i}{f}$. C_j je tzv. normalizovaná kapacita uzlu j (nabývá hodnot z intervalu $\langle 0; 1 \rangle$, N_C značí celkový počet synchronizačních hodinových pulsů během aplikace testu, N_G je celkový počet uzlů v obvodu.

3.1.5 Stanovení metrik pro řetězec scan

V případě použití strategie TPS roste počet potřebných kroků výpočtu výše uvedených metrik m násobně, kde m je počet registrů v řetězci scan. U komplexních obvodů tak může dojít ke značnému prodloužení času výpočtu. Z těchto důvodů byly zavedeny rovnice,

umožňující jednodušeji vyčíslit metriky pro řetězce scan. Nevýhodou těchto přístupů je, že neberou v úvahu překlápění obvodové logiky připojené na scan registry při fázích sériový scan-in/out, čímž se přesnost metriky snižuje. Přesto jsou tyto rovnice dobře použitelné např. v případě scan řetězce s tzv. „izolovatelnými“ registry (viz. kapitola 4). NTC pro scan-in testovacího vektoru aplikovaného přes řetězec scan lze určit pomocí rovnice (3.15), pro scan-out odezvy na testovací vektor pomocí rovnice (3.16), celkové NTC pomocí rovnice (3.17). Tyto rovnice vycházejí z práce [56].

$$NTC_{scin}(V_i) = \sum_{j=1}^{m-1} \left((V_i(j) \; XOR \; V_i(j+1)) \, j \right) \tag{3.15}$$

$$NTC_{scout}(V_i) = \sum_{j=1}^{m-1} \left((V_i(j) \; XOR \; V_i(j+1)) \, (m-j) \right) \tag{3.16}$$

V rovnicích (3.15), (3.16) V_i je testovací vektor o m bitech, který bude načten do řetězce scan složeného z m registrů, V_o je odezva obvodu na testovací vektor V_i.

$$NTC_{sc}(V_i, V_o) = NTC_{scin}(V_i) + NTC_{scout}(V_o) \tag{3.17}$$

Tyto vztahy je možné rozšířit i pro zjednodušený výpočet metrik WNTC (rovnice (3.18)), WSA (rovnice (3.19)) řetězce scan.

$$WNTC_{sc}(V_i, V_o) = NTC_{sc}(V_i, V_o) F_{avg} \tag{3.18}$$

V rovnici (3.18) F_{avg} značí průměrný počet větvení výstupů registrů řetězce scan.

$$WSA_{sc}(V_i, V_o) = NTC_{sc}(V_i, V_o) C_{avg} \tag{3.19}$$

Obdobně v rovnici (3.18) C_{avg} značí průměrnou normovanou kapacitní zátěž výstupů registrů řetězce scan.

3.1.6 Aproximační modely

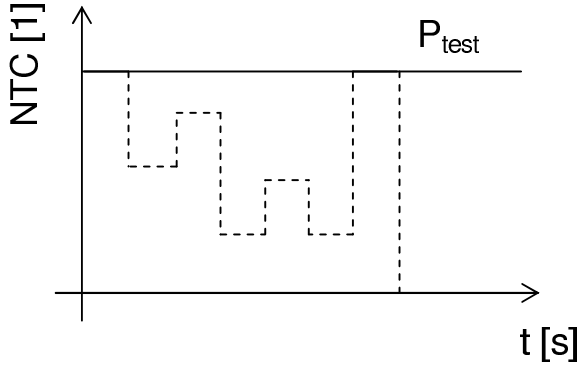
Průběh příkonu obvodu vyžaduje pro uložení a další zpracování vektor reálných čísel o délce rovné počtu synchronizačních pulzů (v případě TPC strategie též počtu testovacích vektorů, viz obr. 3.4). Z hlediska metod plánování testu však není efektivní udržovat pro každý test takto obsáhlou informaci a mnohem výhodnější je použít vhodný způsob aproximace příkonu pro jednotlivé části testu. Na aproximační model jsou kladeny následující požadavky, dle [43]:

Jednoduchost – použitý aproximační model musí být dostatečně jednoduchý, aby jeho praktická implementace nebyla výpočetně příliš náročná.

Bezpečnost – použitý aproximační model musí být bezpečný ve vztahu k reálnému obvodu. Výsledky získané z použitého aproximačního modelu nesmí snižovat tepelné účinky příkonu ve srovnání s reálným obvodem.

Přesnost – použitý aproximační model musí být dostatečně přesný, tedy měl by co nejblíže odpovídat tepelným účinkům příkonu v reálném obvodu bez zbytečného nadhodnocení, jinak by mohlo dojít ke zbytečnému snížení souběžnosti testu a tím i k prodloužení aplikace testu.

Často se používá aproximace průběhu příkonu na určitou skalární hodnotu P_{test}, se kterou lze již dále jednoduše provádět operace porovnání a sčítání. Velmi často bývá hodnota P_{test} stanovena jako maximum z průběhu příkonu obvodu během aplikace testu (obr. 3.5). Takto stanovená hodnota P_{test} je sice bezpečná, protože během aplikace testu nemůže dojít k jejímu překročení, nicméně pro praktické použití je tato hodnota většinou zbytečně nadhodnocena. Smyslem metod plánování testu pro omezení P_{max} je redukce tepelných účinků na obvod, čehož se dosahuje prodloužením aplikace testu. Reálný obvod však má jistou tepelnou setrvačnost, díky čemuž se špičky v průběhu příkonu neprojeví. Při aproximaci P_{test} na maximální hodnotu průběhu příkonu proto může docházet ke zbytečnému prodloužení aplikace testu. V praxi výhodnější tedy může být aproximace P_{test} na střední hodnotu průběhu příkonu. V tomto případě je však třeba vždy prověřit bezpečnost modelu vzhledem k tepelné setrvačnosti použitého obvodu.



Obrázek 3.5: Aproximace P_{test} na maximální hodnotu průběhu příkonu

Kapitola 4

Současný stav řešené problematiky

V této kapitole bude uveden stručný přehled existujících optimalizačních metod pro redukci příkonu během aplikace testu. Tyto metody lze rozdělit do dvou základních skupin – na metody zaměřené na redukci dynamické části příkonu (tzv. switching power) a na metody zaměřené na redukci statické části příkonu (tzv. leakage power). U starších implementačních technologií převažovala dynamická část příkonu nad statickou (např. v [68] je uvedeno, že dynamická složka příkonu tvoří téměř 90 % z celkového příkonu obvodu), díky čemuž se redukci statické části příkonu nevěnovala příliš pozornost. Závislost dynamické části příkonu na napájecím napětí je kvadratická. Jedním z nejefektivnějších způsobů její redukce je proto snižování napájecího napětí. Proto je často zvyšování stupně integrace spojeno se současným snižováním napájecího napětí. Aby byla zachována šumová imunita obvodu, musí být zároveň s napájecím napětím proporcionálně snižována i prahová napětí, díky čemuž dochází k exponenciálnímu nárůstu statické složky příkonu. Proto např. u 90 nm technologie dynamická část příkonu tvoří již jen asi 58 % z celkové hodnoty příkonu (dle [66]). Technologie 65 nm bývá často uvažována jako hranice, kdy statická část příkonu začíná převažovat nad dynamickou. U technologií s ještě vyšším stupněm integrace (32 nm, 25 nm, ...) statická část příkonu již převažuje nad dynamickou [38]. Pro volbu efektivní optimalizační metody je tedy nezbytná znalost cílové implementační technologie.

Další možností rozdělení optimalizačních metod (např. dle [43]) je rozdělení na metody závislé na použitém testu, tzv. TSD metody (Test Set Dependent) a metody nezávislé na použitém testu, tzv. TSI metody (Test Set Independent). První typ metod pro redukci příkonu využívá jak modifikace testu, tak modifikace struktury obvodu. Druhý typ metod pro redukci příkonu využívá pouze modifikace struktury obvodu a redukce příkonu je dosahováno nezávisle na použitém testu. Toto dělení se v literatuře nejčastěji uplatňuje u metod pro redukci dynamické části příkonu, nicméně v případě obecného pohledu lze metody pro redukci statické části příkonu zařadit do kategorie TSI.

Metody pro redukci příkonu během aplikace testu se používají pro omezení negativních jevů způsobených průchodem velkého proudu (elektromigrace, induktivní efekty, pokles napětí na napájecích linkách, ...), pro snížení požadavků na odvod tepla a napájecí zdroj během aplikace testu. Cílem je dosáhnout spolehlivého testu při nízkých nákladech na testování. Dále je možné zavést následující kritérium: Nechť P_1 je střední příkon během aplikace testu před aplikací optimalizační metody a P_2 je střední příkon během aplikace testu po aplikaci optimalizační metody a dále t_1 je doba trvání testu (v sekundách) před aplikací optimalizační metody a t_2 je doba trvání testu po aplikaci optimalizační metody. Pak metody splňující podmínku: $P_1 t_1 > P_2 t_2$ umožňují dosáhnout redukce energie odebrané během aplikace testu. Tyto metody též umožňují prodloužit dobu provozu při napájení z

baterií, či uspořit náklady za energii během testování (např. na výrobní lince). V zahraniční literatuře tyto metody většinou nebývají explicitně rozlišovány.

Mnoho optimalizačních metod pro redukci příkonu je založeno na iteračním principu, kdy je nutné v jednotlivých iteracích metody ověřovat kvalitu dosud dosažených řešení. Důležitá je i možnost přesného porovnání výsledků jednotlivých optimalizačních metod. Z těchto důvodů je nezbytné dále uvést i základní principy metod pro stanovení příkonu obvodu.

4.1 Metody pro stanovení příkonu

Nejspolehlivější metodou je bezesporu přímé měření okamžitých hodnot napětí a odebíraného proudu během aplikace testu a následné vypočtení průběhu příkonu. Přímé měření je však v praxi značně problematické, zejména v případě obvodů pracujících na frekvencích blížících se limitům realizačních technologií. Měření analogovými přístroji je kvůli omezeným možnostem dalšího zpracování nevyhovující. Digitální měřící přístroje musí být zase schopny vzorkovat hodnoty snímaných veličin na vyšší frekvenci než je pracovní frekvence měřených obvodů (minimálně dvojnásobné – Nyquist-Shannonův vzorkovací teorém [61] a v případě požadavku na zachycení průběhu veličin mezi jednotlivými synchronizačními pulzy i vyšší). Měřící přístroje takových parametrů jsou velice nákladné nebo často ani neexistují. Někdy je třeba stanovit příkon jen určitých vnitřních částí obvodu, kde není možné přímé připojení měřících sond. Jednou z možností řešení je nepřímé měření, kdy se využívá např. měření teploty čipu během aplikace testu [5], z jejíž hodnoty lze za pomoci dalších známých parametrů určit příkon obvodu. Může být rozmístěno i více teplotních čidel. Díky tepelné setrvačnosti čipu však tato metoda vede k poměrně nepřesným výsledkům, což znemožňuje provádět přesnější porovnání jednotlivých optimalizačních metod. Dalším problémem je, že v případě aplikace optimalizačních metod modifikujících strukturu obvodu, by bylo třeba vyrábět a měřit velké množství prototypů. To je vzhledem k nákladům (časovým i finančním) na výrobu prototypu nemyslitelné. Z těchto důvodů se častěji pro vyhodnocení výsledku optimalizačních metod používají statistické a simulační metody. Tyto metody většinou využívají různých zjednodušujících metrik. Pro účely porovnaní kvality řešení u metod sloužících k redukci dynamické části příkonu se jeví jako dostačující metrika NTC. Pro kvalitnější porovnání bývají aplikovány i jiné přesnější metriky (WNTC, WSA, ...). Pro nejvyšší přesnost simulace nebo v případě nezanedbatelné velikosti statické části příkonu je nutné během simulace pracovat s okamžitou hodnotou příkonu P, což je však výpočetně velice náročné.

Statistické metody stanovení příkonu se vyznačují nízkou výpočetní složitostí (vysokou rychlostí), avšak jsou nejméně přesné ([49]). Tyto metody pracují s údaji, jako je typ a počet obvodových prvků, průměrný počet rozvětvení v obvodu, délka řetězce scan, atp. Simulační metody dosahují vyšší přesnosti. Simulační metody je možné rozdělit např. dle [49] na metody využívající plné syntézy (simulace na fyzické úrovni), metody využívající omezené syntézy a tzv. metodu černých skříněk. V případě simulace využívající plné syntézy probíhá simulace přímo na úrovni rozvržení čipu. Tato simulace je nejpřesnější, ovšem také nejvíce časově náročná. V případě simulace využívající omezené syntézy, se návrh obvodu namapuje na definovanou množinu prvků (tzv. technologickou knihovnu). V této knihovně jsou k dispozici modely, ze kterých je možné získat požadovaná data s dostatečnou přesností. Tyto modely jsou většinou vytvořeny pomocí simulace na fyzické úrovni návrhu a případně i ověřeny měřením. Metoda černých skříněk zase seskupuje vybrané části obvodu do bloků, tzv. černých skříněk. Nad těmito bloky jsou natrénovány odezvy na definovaná vstupní data.

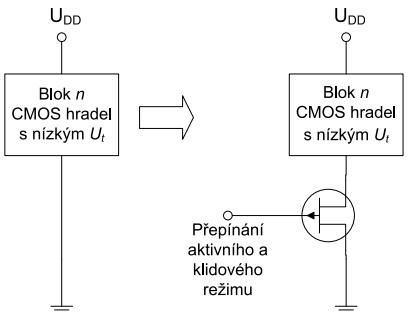
Během simulace se pro získání požadovaných výsledků využívá interpolace/extrapolace z natrénovaných odezev.

4.2 Metody redukce příkonu nezávislé na použitém testu

4.2.1 Metody pro redukci statické části příkonu

Tyto metody se uplatňují zejména u nejnovějších technologií s nejvyšším stupněm integrace. Jsou vhodné pro technologie $\leq 90\,\text{nm}$, jejichž zastoupení na trhu stále roste (dle [74] v roce 2006 10 % všech začínajících návrhů používalo technologii $\leq 90\,\text{nm}$).

Jednou z nejrozšířenějších metod pro redukci statické části příkonu je metoda MTCMOS [6]. Metoda je založena na vkládání tzv. „footer“ tranzistorů do „pull-down“ sítě u CMOS bran. Přídavný tranzistor slouží k odpojení CMOS brány v klidovém režimu, díky čemuž dochází k redukci statické složky příkonu. Nevýhodou metody je zhoršení dynamických vlastností obvodu a navýšení plochy čipu, protože každá CMOS brána vyžaduje samostatný přídavný tranzistor. Tyto nevýhody se snaží odstranit metoda PG (Power Gating) [37]. U této metody je přídavný tranzistor použit k odpojování celé skupiny hradel (tzv. blok) místo jednotlivých hradel viz obr. 4.1.



Obrázek 4.1: Princip PG metody

Ačkoliv princip PG metody se zdá být jednoduchý, její efektivní implementace není triviální. Je třeba zvolit vhodnou velikost přídavného tranzistoru, protože ten výrazně ovlivňuje parametry všech hradel k němu připojených. Též je nutné vhodně zvolit granularitu bloků, které budou odpojovány pomocí přídavného tranzistoru. Dalším vylepšením je metoda CPG (Clustered PG) [58], u které může být každý blok tvořen libovolným počtem hradel, případně mohou existovat i hradla nepřiřazená k blokům. Cílem CPG metody je identifikovat v obvodu tyto bloky.

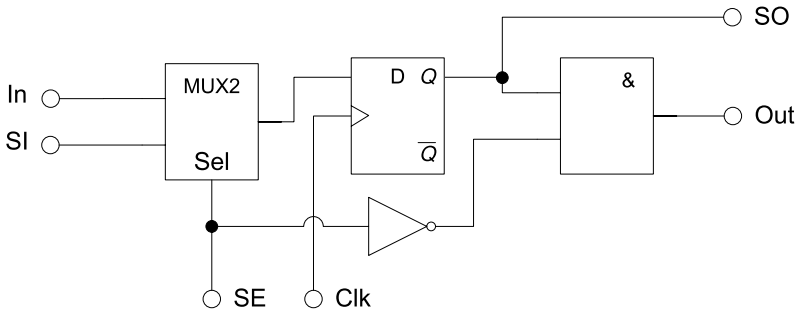
Jiný přístup lze nalézt v [54]. Zde publikovaná metoda hledá optimální prahové napětí U_t pro obvodová hradla, při které nebude překročeno povolené zpoždění obvodu. Využívá se skutečnosti, že hodnota U_t kromě příkonu hradla ovlivňuje i jeho zpoždění (viz (3.11)). Metoda začíná s nejnižším možným U_t, které je postupně zvyšováno pro všechna hradla v obvodu. Zvýšení U_t ovlivní zpoždění obvodu, proto je po každém kroku přepočítána kritická cesta (cesta s největším zpožděním) v obvodu. Tento postup se opakuje, dokud zpoždění obvodu nedosáhne povolené maximální hodnoty. Výstupem metody je obvod s výrazně redukovanou statickou složkou příkonu, jehož zpoždění nepřekračuje povolenou maximální hodnotu.

4.2.2 Metody pro redukci dynamické části příkonu

Mezi TSI metody pro redukci dynamické části příkonu lze zařadit i nejrůznější metody pro syntézu nízkopříkonových obvodů [59]. Jejich aplikací dochází sice k redukci příkonu, nicméně disproporci mezi příkony v diagnostickém a funkčním režimu činnosti obecně neodstraní. V praxi používanou metodou redukce dynamické části příkonu je metoda využívající dvojitého napájecího napětí [67]. Metoda identifikuje kritickou cestu v obvodu. Prvky obvodu, které leží na kritické cestě jsou pak napájeny vyšším napětím, zatímco prvky ležící mimo kritickou cestu jsou napájeny nižším napětím. Tímto způsobem je možné dosáhnout redukce příkonu při zachování zpoždění obvodu. Další metodou využívající kritické cesty je [25]. Tato metoda pracuje s hradly rozdílné velikosti. Hradla ležící na kritické cestě jsou zvětšena, zatímco ostatní hradla jsou zmenšena. Výsledný efekt je srovnatelný s předchozí metodou. Další možností je použít více než jednoho synchronizačního signálu, každý o jiné frekvenci. Tento přístup bývá někdy pro vylepšení výsledků kombinován s použitím více napájecích napětí. Na tomto kombinovaném principu je založena např. metoda [51].

Dále budou uvedeny metody, které jsou přímo zaměřeny na odstranění disproporce mezi příkony v diagnostickém a funkčním režimu činnosti. Velice rozšířena je metoda využívající variabilní frekvence synchronizačního signálu [69]. U této metody je během aplikace testu frekvence synchronizačního signálu snížena, díky čemuž je dosaženo redukce příkonu. Nevýhodou je, že je obvod testován na nižší rychlosti než je provozní, díky čemuž nemusí být odhaleny dynamické chyby časování způsobené např. parazitními kapacitami a aplikace testu trvá déle.

Další metody se zaměřují na modifikace řetězce scan. U obvodů obsahujících řetězec scan dochází během sériového načítání/vyčítání testovacích vektorů/odezev ke zbytečnému překlápění kombinační logiky připojené k registrům scan. Tomu se snaží zabránit metoda využívající izolovatelných scan registrů [19]. Nejjednodušší způsob vytvoření izolovatelného scan registru ze standardního scan registru je přidání blokujícího hradla (obr. 4.2). Tato úprava zamezí překlápění logiky připojené na tento scan registr během sériového načítání/vyčítání dat přes scan řetězec. Na obr. 4.2 SI označuje vstup pro připojení k předchozímu scan registru (Scan In), SO je výstup pro připojení na následující scan registr (Scan Out), In je vstup pro snímaní odezvy testovaného obvodu, Out je výstup pro aplikaci testovacích vektorů. Pomocí vstupu SE je možno přepínat mezi funkcí scan registru (hodnota logická „1“) a normálního registru (logická hodnota „0“).



Obrázek 4.2: Izolovatelný scan registr

Nevýhodou metody je nárůst množství přídavné obvodové logiky a prodloužení zpoždění ve funkčním režimu činnosti obvodu. Jiný přístup využívá metoda popsaná v [8], která redukuje příkon snížením frekvence synchronizačního signálu u scan registrů. Řetězec scan

je rozdělen na dvě poloviny. Aby nedošlo k ovlivnění dynamických vlastností obvodu, využívá se dvou fázově posunutých synchronizačních signálů Clk_A, Clk_B, každý o poloviční frekvenci oproti původnímu signálu Clk. Synchronizace Clk_A je přiváděna na první polovinu řetězce scan a synchronizace Clk_B je přiváděna na druhou polovinu řetězce scan, díky čemuž zůstává rychlost řetězce scan zachována. Pokud je k dispozici dostatek primárních vstupů/výstupů (případně je možné je pro diagnostický režim uvolnit, např. vložením multiplexoru) je další možností redukce příkonu použití více scan řetězců. Tohoto principu je využito např. u metody [72]. Metoda popsaná v [36] tento princip dále rozšiřuje a pracuje s dvourozměrným pole scan registrů. Naproti tomu metoda definovaná v [75] využívá prokládaného řetězce scan s přídavným zpožděním, což umožňuje dosáhnout redukce špičkového příkonu během paralelního načítání odezvy obvodu na testovací vektory.

4.3 Metody redukce příkonu závislé na použitém testu

V [71] je představena metoda využívající speciálního ATPG algoritmu, který zvyšuje korelaci mezi generovanými testovacími vektory. Nevýhodou této metody je prodloužení aplikace testu. Metoda popsaná v [31] využívá tzv. „don't care“ bitů, které nastavuje na hodnoty logická „0“ nebo „1“ tak, aby byla redukována přepínací aktivita v obvodu. Při testování pomocí pseudonáhodných vektorů lze použít i metodu eliminace testovacích vektorů [20]. Vektory, které výrazněji nepřispívají k pokrytí chyb, jsou z testu vyřazeny, čímž je dosaženo zkrácení testu a úspory energie. Metody založené na tomto principu nalézají největšího uplatnění v případě BIST, kde se LFSR doplní o dekodér, jenž je navržen pro detekci a vyřazení testovacích vektorů odpovídajících předem stanovené masce. Dále popsané metody jsou zaměřeny na řetězec scan. V [56] byl představen algoritmus pro převod scan vektorů na novou kompaktnější formu tak, aby nedošlo k překročení předem definovaného maximálního příkonu. V [57] byla popsána metoda využívající specifického přístupu k aplikaci testu s využitím více řetězců scan. Pro účely testu je pro každý řetězec scan přidán pomocný signál „scan chain disable“, který umožňuje odpojení scan registrů daného řetězce od synchronizačního signálu. Metoda využívá myšlenky, že v případě většího množství scan řetězců je možné identifikovat fáze, ve kterých nejsou určité scan řetězce využity a mohou být odpojeny pro redukci příkonu.

V případě scan řetězce příkon ovlivňuje i použitá strategie testu – tedy, ve které části scan cyklu dochází k nastavení testovacích vektorů na primární vstupy obvodu. Dle [43] lze rozlišit dvě základní strategie testu: ASAP (As Soon As Possible – nastavení testovacích vektorů co nejdříve) a ALAP (As Last As Possible – nastavení testovacích vektorů co nejpozději). Příklad ASAP i ALAP strategií testu pro tři scan cykly u obvodu se třemi scan registry lze vidět ve druhém a třetím sloupci tabulky 4.1.

V tabulce 4.1 první sloupec specifikuje jednotlivé synchronizační pulzy. Pro načtení scan vektoru, jeho aplikaci a vyčtení odezvy obvodu na předchozí scan vektor, je v případě tří scan registrů potřeba čtyř synchronizačních pulzů. V případě ASAP strategie je testovací vektor nastaven na primární vstupy obvodu vždy před příchodem prvního synchronizačního pulzu daného cyklu scan řetězce, jak lze vidět ve třetím sloupci tabulky. Naproti tomu v případě ALAP strategie je testovací vektor nastaven na primární vstupy obvodu vždy před příchodem posledního synchronizačního pulzu (viz čtvrtý sloupec tabulky 4.1). Volba vhodné strategie testu může značným způsobem ovlivnit počet překlopení v obvodu během aplikace testu. Dalším vylepšením je strategie BPIC (Best Primary Input Change time – nastavení testovacích vektorů v nejvýhodnější dobu). Příklad strategie BPIC lze vidět v pátém sloupci tabulky 4.1. V [43] byl definován BPIC algoritmus pro výpočet nejvýhodnějších

Tabulka 4.1: Příklady strategií testu ASAP, ALAP, BPIC.

		ASAP	**ALAP**	**BPIC**	**ETV**
Synchronizační pulzy	Scan cyklus	PI Test vektor	PI Test vektor	PI Test vektor	PI Test vektor
t_1	S_1	V_1	–	–	E_1
t_2	S_1	V_1	–	V_1	E_2
t_3	S_1	V_1	–	V_1	E_3
t_4	S_1	V_1	V_1	V_1	V_1
t_5	S_2	V_2	V_1	V_1	E_4
t_6	S_2	V_2	V_1	V_1	E_5
t_7	S_2	V_2	V_1	V_1	E_6
t_8	S_2	V_2	V_2	V_2	V_2
t_9	S_3	V_3	V_2	V_2	E_7
t_{10}	S_3	V_3	V_2	V_3	E_8
t_{11}	S_3	V_3	V_2	V_3	E_9
t_{12}	S_3	V_3	V_3	V_3	V_3

časů nastavení testovacích vektorů. Na několika jednoduchých příkladech publikovaných společně s definicí algoritmu bylo s BPIC dosaženo redukce NTC 90–95 % z původní hodnoty ASAP/ALAP. Další možností redukce příkonu je použití tzv. dodatečných vektorů. Tento princip byl využit např. v [28]. Principiálně se vlastně jedná o strategii ALAP, u které je před příchodem každého synchronizačního pulzu daného scan cyklu kromě posledního na primární vstupy obvodu nastaveny dodatečné vektory. Tyto vektory jsou sestaveny tak, aby bylo omezeno překlápění obvodové logiky připojené ke scan registrům během sériového načítání/vyčítání dat. Metoda bývá často označována zkratkou ETV (Extra Test Vectors – dodatečné testovací vektory). Příklad metody ETV lze vidět v posledním sloupci tabulky 4.1. V tomto příkladu jsou písmenem „E“ označeny dodatečné testovací vektory a písmenem „V“ normální testovací vektory. Nevýhodou metody je, že zvyšuje objem diagnostických dat, což klade vysoké nároky na paměť ATE, které je většinou nedostatek i bez použití ETV. Např. dle [44] v některých případech je nutné test některých komplexních VLSI obvodů rozdělit na více částí, protože se celý nevejde do paměti běžně dostupných v ATE.

4.3.1 Optimalizace pořadí aplikace testovacích vektorů a zapojení scan registrů

V dnešní době lze pomocí komerčních nástrojů generovat kvalitní sady testovacích vektorů s vysokým stupněm pokrytí chyb. Generované testovací vektory však ve většině případů nejsou optimalizovány pro nízký příkon. Z tohoto důvodu jsou populární různé optimalizační metody, zaměřené na optimalizaci dříve vygenerovaných testů. U kombinačních obvodů je odezva obvodu závislá pouze na aktuálně použitém testovacím vektoru, takže je možné libovolně měnit pořadí aplikace testovacích vektorů a odezvy obvodu na testovací vektory budou stále stejné, pouze v jiném pořadí. Seřazením testovacích vektorů do vhodné posloupnosti je tak možné minimalizovat počet překlopení v obvodu, aniž by došlo ke snížení pokrytí chyb testem. Avšak u sekvenčních obvodů nelze tento přístup přímo použít, protože u nich odezva obecně závisí nejen na aktuálně aplikovaném testovacím vektoru, ale i na po-

Tabulka 4.2: Optimalizace pořadí vektorů a registrů v řetězci scan.

v/SC	SC_1	SC_2	SC_3	SC_4			SC_2	SC_3	SC_4	SC_1
vi_1	1	1	0	0		vi_3	0	1	1	0
vo_1	1	0	0	1		vo_3	1	1	0	0
vi_2	0	1	1	0	$\Rightarrow$	vi_1	1	0	0	1
vo_2	0	0	0	0		vo_1	0	0	1	1
vi_3	0	0	1	1		vi_2	1	1	0	0
vo_3	0	1	1	0		vo_2	0	0	0	0
	$NTC_{orig} = 30$						$NTC_{opt} = 20$			

sloupnosti testovacích vektorů aplikovaných v předchozích krocích. Změnou posloupnosti aplikace testovacích vektorů u těchto obvodů zpravidla získáme úplně jiný test, bohužel většinou s naprosto nevyhovujícím pokrytím chyb. Sekvenční obvody lze však modifikovat vložením plného řetězce scan, potom se obvody pro účely testu jeví jako kombinační a pokrytí chyb testem nezávisí na pořadí aplikace testovacích vektorů, ovšem za cenu nárůstu plochy čipu. Dále je také možné použít optimalizaci pořadí zapojení registrů do řetězce scan. Jednoduchý příklad demonstrující účinnost tohoto kombinovaného přístupu lze vidět v tabulce 4.2.

Testovací vektory mohou být aplikovány přes primární vstupy obvodu nebo přes scan řetězec (řetězce). V případě, že bude nutné dále v textu tyto dva typy testovacích vektorů rozlišit, tak bude explicitně používán termín „scan vektory“ pro označení testovacích vektorů aplikovaných přes scan řetězec (řetězce). V tabulce 4.2 jsou pro zjednodušení uvažovány pouze scan vektory.

V tabulce 4.2 řádky vi_y ($y = 1 \ldots 3$) představují jednotlivé scan vektory a řádky vo_z ($z = 1 \ldots 3$) představují jednotlivé odezvy obvodu na odpovídající scan vektory vi, které budou vyčítány pomocí řetězce scan. Sloupce SC_x ($x = 1 \ldots 4$) představují jednotlivé scan registry. Levá část tabulky 4.2 představuje test, pro který bude $NTC = 30$. Pravá část tabulky 4.2 představuje optimalizovaný test s $NTC = 20$. Oba testy mají shodné pokrytí chyb. Optimalizovaný test byl získán změnou pořadí aplikace scan vektorů a změnou pořadí zapojení registrů do řetězce scan. Prezentovaná optimalizace představuje aplikaci scan vektorů v pořadí 3, 1, 2 a zapojení scan registrů v pořadí 2, 3, 4, 1. NTC optimalizovaného testu bude tedy 66,7 % původní hodnoty NTC, jak je vidět po vyčíslení hodnoty koeficientu r (redukce) v rovnici (4.1). Způsob stanovení NTC pro levou část tabulky 4.2 je zobrazen v tabulce 4.3.

$$r = \frac{NTC_{opt}}{NTC_{orig}} 100 = \frac{20}{30} 100 = 66,7\% \tag{4.1}$$

V tabulce 4.3 je rozepsána aplikace testu z levé části tabulky 4.1. První sloupec tabulky představuje jednotlivé synchronizační pulzy. Test začíná synchronizačním pulzem číslo 1. Na prvním řádku tabulky je v prvním sloupci uvedeno 0, protože tento řádek představuje definici počátečního stavu. Pro stanovení NTC je v tomto případě uvažován počáteční i koncový stav s hodnotami logická 0. V tomto příkladu není kvůli zjednodušení uvažována vnitřní struktura obvodu. V praxi však každé překlopení $0 \rightarrow 1$, $1 \rightarrow 0$ v tabulce 4.3 způsobí množství vnitřních překlopení v obvodu, které je nutné odsimulovat a započíst. Během synchronizačních pulzů 1 až 4 dochází k sériovému načítání scan vektoru vi_1. Řetězec scan je složen ze čtyřech scan registrů, proto je třeba k sériovému načtení vektoru čtyř

Tabulka 4.3: Příklad stanovení NTC.

i/SC	SC_1	SC_2	SC_3	SC_4	Poznámka
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	
3	1	0	0	0	
4	**1**	**1**	**0**	**0**	vi_1
5	**1**	**0**	**0**	**1**	vo_1
6	0	1	0	0	
7	1	0	1	0	
8	1	1	0	1	
9	**0**	**1**	**1**	**0**	vi_2
10	**0**	**0**	**0**	**0**	vo_2
11	1	0	0	0	
12	1	1	0	0	
13	0	1	1	0	
14	**0**	**0**	**1**	**1**	vi_3
15	**0**	**1**	**1**	**0**	vo_3
16	0	0	1	1	
17	0	0	0	1	
18	0	0	0	0	
19	0	0	0	0	
$NTC =$	**6**	**+ 10**	**+ 6**	**+ 8**	**= 30**

synchronizačních pulzů. Po čtvrtém synchronizačním pulzu bude tedy ve scan řetězci načten vektor vi_1, jak lze vidět v tabulce. Po příchodu pátého synchronizačního pulzu dochází k aplikaci scan vektoru vi_1 a do scan řetězce je paralelně načtena odezva obvodu vo_1, která je během synchronizačních pulzů 6 až 9 sériově vyčítána ze scan řetězce. Zároveň je sériově načítán nový scan vektor vi_2. Po příchodu desátého synchronizačního pulzu dochází k aplikaci scan vektoru vi_2 a do scan řetězce je paralelně načtena odezva obvodu vo_2. Pro scan vektor vi_3 je princip shodný. Další scan vektory již nejsou k dispozici, takže během synchronizačních pulzů 16 až 19 dochází pouze k vyčtení odezvy vo_3 ze scan řetězce, čímž je aplikace testu ukončena. Jakmile máme sestavenou tabulku 4.3 lze přistoupit k vlastnímu výpočtu NTC. Výpočet spočívá v sečtení přechodů $0 \rightarrow 1$, $1 \rightarrow 0$ v jednotlivých sloupcích tabulky. Pro první sloupec tabulky tedy dostáváme hodnotu 6, pro druhý sloupec tabulky hodnotu 10, pro třetí sloupec tabulky hodnotu 6, pro čtvrtý sloupec tabulky hodnotu 8. Sečtením těchto hodnot dostáváme výslednou hodnotu NTC, což je v tomto případě 30. Výpočet NTC pro pravou část tabulky 4.1 je analogický a nebude tedy uváděn.

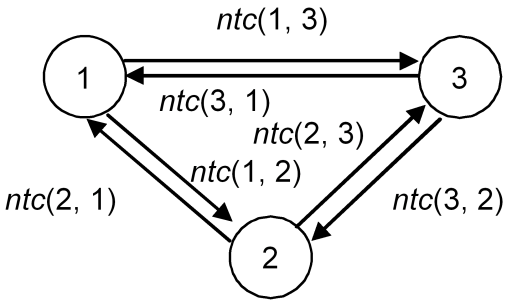
Problém nalezení vhodné posloupnosti testovacích vektorů/scan registrů patří do kategorie „NP-složitých“ problémů (NP-hard problems) [14]. Problém má obecně složitost $O(n) = n!$, kde n je počet prvků, jejichž pořadí má být optimalizováno. Pro modelování problému se často využívá grafového modelu, který je možné použít pro scan registry i pro testovací vektory. V obou případech je princip shodný, nicméně oba přístupy nelze modelovat současně pomocí jednoho grafového modelu. Grafový model pro problém optimalizace pořadí aplikace testovacích vektorů pro nízký příkon lze definovat pomocí definice 4.3.1.

Definice 4.3.1.

- *Nechť V je množina všech vrcholů grafu, VI je množina všech testovacích vektorů, platí $|V| = |VI|$ a existuje bijekce $V \leftrightarrow VI$,*
- *nechť E je množina všech hran grafu pro kterou platí $E \subseteq V \times V$,*
- *nechť existuje funkce ohodnocující jednotlivé hrany grafu $ntc : E \rightarrow \aleph$ zapisovaná jako $ntc(vi_a, vi_b)$, která definuje počet překlopení v obvodu při aplikaci testovacího vektoru vi_b po vi_a,*

potom orientovaný ohodnocený graf $G = (V, E)$ modeluje problém optimalizace pořadí aplikace testovacích vektorů pro nízký příkon.

Příklad tímto způsobem vytvořeného grafu pro obvod, jehož test spočívá v aplikaci tří testovacích vektorů $VI = 1, 2, 3$, lze vidět na obrázku 4.3. Další postup pak spočívá v hledání minimální Hamiltonovy cesty v takto vytvořeném grafu. Minimální Hamiltonova cesta odpovídá hledané posloupnosti aplikace testovacích vektorů, pro kterou je střední hodnota příkonu během aplikace testu minimální. Hledání minimální Hamiltonovy cesty je známý „NP-složitý“ problém.



Obrázek 4.3: Grafový model problému

Existuje množství metod využívajících výše uvedeného přístupu. Např. pouze na optimalizaci pořadí testovacích vektorů je zaměřena metoda popsaná v [21]. Pro stanovení příkonu využívá výpočtu Hammingovy vzdálenosti mezi testovacími vektory. [21] dále porovnává tento přístup se simulací pomocí nástrojů ze sady HITEST u vybraných obvodů a konstatuje, že příkon během aplikace testu má spojitost s Hammingovou vzdáleností mezi jednotlivými testovacími vektory. V kapitole 8 této práce však bude prokázáno, že existují případy, ve kterých výsledky nebudou korelovat. Bez znalosti způsobu fyzické realizace obvodu je totiž problematické usuzovat na přepínací aktivitu uvnitř obvodu. Změna jednoho bitu tak může způsobit i větší počet překlopení než změna několika jiných bitů. V [11] je využito pouze optimalizace pořadí registrů v řetězci scan. Pro prohledávání stavového prostoru se využívá metody „greedy search“ (hladový algoritmus).

Existují i metody kombinující BPIC strategii testu se změnou pořadí testovacích vektorů, díky čemuž mohou dosahovat vyšší redukce příkonu. Např. v [43] je popsána metoda založená na tomto kombinovaném přístupu, která pro prohledání stavového prostoru využívá simulovaného žíhání. Nevýhodou těchto metod však je, že vyžadují specifický přístup k aplikaci testu, takže jejich nasazení ve spojení s komerčními diagnostickými nástroji je problematické.

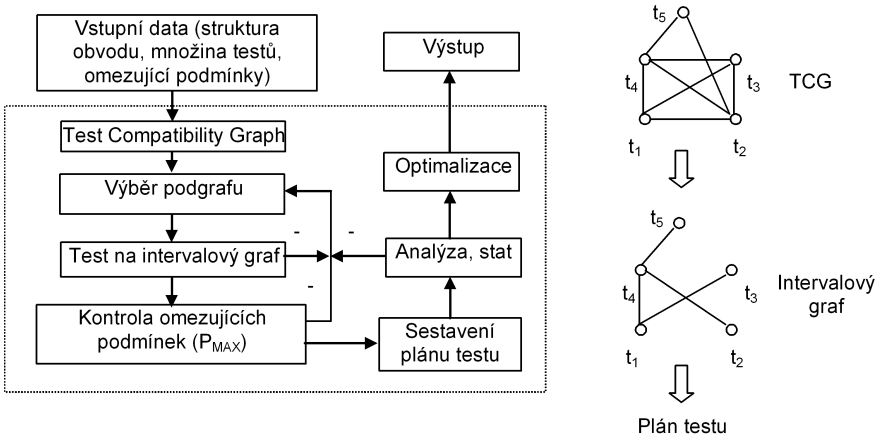
V některých případech se používá postupné aplikace jednotlivých optimalizačních metod (např. aplikace metody pro optimalizaci posloupnosti testovacích vektorů po metodě optimalizující posloupnost zapojení registrů v řetězci scan). Mezi oběma těmito optimalizacemi však existují vzájemné závislosti (první metoda ovlivní výsledky druhé), takže tímto způsobem nelze dosáhnout nejkvalitnějších výsledků. Pro dosažení lepších výsledků je nutné oba přístupy kombinovat již při prohledávání stavového prostoru.

4.3.2 Metody plánování testu

Metody plánování testu je možno principiálně zařadit do kategorie TSD. Existuje množství metod zabývajících se plánováním testu pro nízký příkon, např. [10; 12; 29; 35; 52; 60; 65]. Tyto metody jsou určeny zejména pro SoC, kdy je na obvod nahlíženo na vyšší úrovni abstrakce. Při této úrovni pohledu je možné identifikovat jednotlivé funkční bloky, či přímo IP-jádra. Cílem těchto metod je naplánovat test takovým způsobem, aby bylo efektivně využito všech dostupných zdrojů (vodičů, sběrnic, scan řetězců) pro test obvodu a byla minimalizována doba aplikace testu a zároveň, aby během aplikace testu nedošlo k překročení maximálního povoleného příkonu. Obecný problém plánování testu je „NP-složitý“ problém, což dokázal Chakrabarty v [10]. Proto se pro řešení tohoto problému využívá různých zjednodušení nebo přístupu založených na heuristice.

V [29] byla popsána metoda založená na preemptivním a prioritním plánování testu. Tato metoda mimo jiné plánuje test takovým způsobem, aby byly dříve testovány bloky, které jsou více náchylné k selhání (předpokládá se, že se test po první nalezené chybě zastaví), čímž je možné dosáhnout zkrácení testu u poruchových obvodů. Ovšem vzhledem k předpokladu, že v praxi většina testovaných obvodů nebude vykazovat poruchu, není tato vlastnost velkou výhodou. V [35] byla představena metoda, využívající „greedy search“ algoritmu pro paralelizaci testu s ohledem na příkon. V přístupu prezentovaném v [12] se používá graf zdrojů pro modelování testovaného obvodu a z něj se vytváří TCG (Test Compatibility Graph). Plán testu je získán řešením problému minimálního pokrytí grafu. Další metodou založenou na analýze TCG je např. metoda popsaná ve [65]. Metoda převádí TCG na intervalový graf, ve kterém jsou potom vyhledávány kliky. Vyhledávání klik probíhá iteračně. Při každé iteraci je odebrána náhodně zvolená hrana grafu, který je následně testován, zda má vlastnosti intervalového grafu. Tímto způsobem jsou ze stavového prostoru úlohy vyřazeny nevyhovující grafy. Dle nalezené kliky grafu je potom sestaven vlastní plán testu. Pro prohledávání stavového prostoru je použita metoda Tabu search. Princip metody lze vidět na obr. 4.4.

Metoda využívající grafu TACG (Test Application Conflict Graph), tedy grafu inverzního k TCG, byla představena v [7]. V této původní verzi ovšem metoda neumožňovala definovat omezení v podobě maximálního povoleného příkonu. V rámci řešení tématu disertační bylo provedeno její rozšíření o možnost omezení maximálního povoleného příkonu Takto modifikovaná metoda byla publikována v [77].



Obrázek 4.4: Metoda plánování testu založená na Tabu search

Kapitola 5

Cíle disertační práce

V dnešní době je návrhářům k dispozici množství automatizovaných návrhových prostředků, které umožňují kvalitní strukturovaný návrh číslicových obvodů. Tyto návrhové prostředky již obsahují specializované nástroje pro zajištění dobré testovatelnosti obvodu. Zpravidla se jedná o nástroje umožňující do návrhu obvodu vkládat pomocné diagnostické prostředky jako jsou řetězce scan, rozhraní JTAG, jednotky pro dekompresi testovacích vektorů, apod. V případě řetězců scan bývá při průmyslovém návrhu upřednostňován plný scan před částečným scanem zejména kvůli přímočarosti implementace i kvůli následné jednodušší aplikaci testu, přestože je pro implementaci třeba větší plocha čipu.

Pro obvody obsahující plný řetězec scan umožňují moderní generátory testu generovat testovací vektory s vysokým pokrytím chyb, avšak příkon obvodu během aplikace testu nebývá při generování těchto testovacích vektorů zohledňován. Vzhledem k tomu, že příkon obvodu během aplikace testu je obecně vyšší než při běžném funkčním režimu činnosti obvodu, otevírá se zde prostor pro výzkum. Pro obvody CMOS bylo dosud publikováno množství metod pro redukci příkonu i pro redukci odebrané energie během aplikace testu. Pro obvody s plným řetězcem scan se jako zajímavé jeví optimalizace založené na principu reorganizace pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce scan. Tyto optimalizace lze principiálně snadno začlenit do existujícího návrhového procesu, takže je možné dále využívat stávající návrhové prostředky. Aplikace těchto optimalizací je však problematická kvůli obrovské velikosti stavového prostoru této optimalizační úlohy. Pro většinu běžných obvodů nepřipadá kompletní prohledávání stavového prostoru úlohy v úvahu. Většina dosud publikovaných metod proto využívá nejrůznějších heuristik. Často se pro vylepšení výsledků jednotlivé optimalizace kombinují a jsou aplikovány postupně za sebou, tedy např. nejdříve se aplikuje optimalizace pořadí aplikace testovacích vektorů a následně se aplikuje optimalizace pořadí zapojení registrů do řetězce scan, případně v opačném pořadí. Takto však může při prohledávání stavového prostoru dojít k přehlédnutí některých zajímavých řešení, protože obě optimalizace jsou vzájemně závislé, nehledě na delší dobu nutnou pro nalezení řešení. Dalším problémem je vyhodnocení kvality řešení. Vzhledem k počtu zkoumaných řešení je třeba pro ohodnocení kvality jednotlivých řešení použít co nejrychlejší metodu. Na přesnosti ohodnocení jednotlivých řešení totiž přímo závisí velikost dosažené redukce příkonu. Většina dosud publikovaných metod využívá nejrůznější zjednodušení pro rychlé ohodnocení kvality řešení. Často se využívá výpočtu Hammingovy vzdálenosti mezi testovacími vektory, dle které se usuzuje na příkon obvodu. Tento výpočet je sice velice rychlý a snadno implementovatelný, avšak získané výsledky nejsou příliš přesné, protože příkon obvodu obecně závisí jak na vstupních datech, tak na struktuře obvodu. Ze změny jednotlivých bitů vstupního vektoru bez znalosti vnitřní struktury obvodu

je totiž problematické usuzovat na počet překlopení uvnitř obvodu. Stejné Hammingovy vzdálenosti tak mohou odpovídat naprosto rozdílným příkonům, což se musí jistě projevit na kvalitě dosažených výsledků při aplikaci této metriky.

5.1 Cíle

1. Hlavním cílem disertační práce je navrhnout a implementovat metodiku pro redukci příkonu obvodu během aplikace testu využívající současné optimalizace pořadí aplikace testovacích vektorů a zapojení registrů do řetězce scan. Pro splnění tohoto hlavního cíle je nutné splnit následující dílčí cíle:

   (a) Rozšířit existující formální model číslicového obvodu. Nad tímto modelem bude metodika vystavěna.

   (b) Sestavit knihovnu pro technologii AMI popisující vlastnosti jednotlivých obvodových prvků. Knihovna bude využita při simulaci aplikace testu u obvodů syntetizovaných do technologie AMI.

   (c) Navrhnout a implementovat simulační metodu pro stanovení příkonových metrik (NTC, WNTC, WSA, P) obvodu během aplikace testu s využitím vytvořené knihovny.

   (d) Aplikovat genetický algoritmus na prohledávání stavového prostoru úlohy. Navrhnout způsob kódování optimalizačního problému do genotypu, definovat algoritmy pro převod genotypu na fenotyp a opačně.

2. Provést experimentální ověření implementované metodiky a porovnání s existujícími metodami.

Kritéria řešení lze definovat následovně: Metodika musí umožnit redukci střední hodnoty příkonu i energie odebrané během aplikace testu. Metodika musí být použitelná pro obvody obsahující plný řetězec scan (tvořený jedním nebo více řetězci scan). Optimalizace pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce (řetězců) scan musí být možné aplikovat souběžně, což dosud publikované metody neumožňují. Metodika musí též podporovat i samostatnou optimalizaci pořadí aplikace testovacích vektorů v případě kombinačních obvodů. U libovolného obvodu nesmí být po aplikaci metodiky energie odebraná během aplikace testu vyšší než před aplikací metodiky. Metodika musí být dostatečně robustní, aby ji bylo možné aplikovat i na komplexní VLSI číslicové obvody. Při ohodnocování kvality jednotlivých řešení při výpočtu fitness musí být dosaženo dostatečné přesnosti, předpokládá se dosažení vyšší přesnosti než u existujících metod. Vytvořená implementace metodiky musí být použitelná s moderními návrhovými systémy a přímo zařaditelná do návrhového procesu.

Kapitola 6

Návrh metodiky

Navržená metodika umožňuje u analyzovaného číslicového obvodu (CUA - Circuit under Analysis) **provádět současnou optimalizací pořadí aplikace testovacích vektorů i zapojení registrů do řetězců scan**, čímž lze dosáhnout vyšší redukce příkonu, než když jsou obě optimalizace provedeny nezávisle na sobě (bude prokázáno v kapitole 8). Využívá se simulace aplikace testu nad technologickou knihovnou. Technologická knihovna obsahuje logické, elektrické i fyzikální charakteristiky standardních obvodových prvků. Během simulace se využívá těchto informací pro stanovení příkonových metrik. V rámci práce byly vytvořeny knihovny pro technologie AMI $0,5\,\mu m$, AMI $1,2\,\mu m$ ve variantách *slow*, *typ*, *fast*. Popis formátu knihovny byl zveřejněn na URL ([76]), takže je možné přidat i další technologie.

V kapitole 6.1 je popsán způsob aplikace testu tak, jak bude uvažován při návrhu metodiky. Různé profesionální návrhové systémy totiž přistupují k aplikaci testu rozdílným způsobem. Z tohoto důvodu budou definovány pojmy *kompatibilní test* a *nekompatibilní test*. Toto označení je dále použito pro rozlišení způsobů aplikace testu, které jsou kompatibilní/nekompatibilní s navrženou metodikou. Nekompatibilní testy musí být před aplikací metodiky upraveny na kompatibilní tvar. Způsob úpravy pro několik nejčastěji se vyskytujících nekompatibilních způsobů aplikace testu je uveden v kapitole 6.2. Kapitola 6.3 slouží jako úvod do problematiky genetických algoritmů (GA) . Jsou definovány základní pojmy z oblasti GA, jež jsou nezbytné pro návrh metodiky. Základní princip metodiky je obecně vysvětlen v kapitole 6.4. V kapitole 6.5 je definován formální model, nad kterým bude metodika vystavěna. V kapitole 6.6 jsou definovány operátory a pomocné algoritmy, které budou použity při definici algoritmů pro kódování/dekódování problému do/z genotypu jedinců populace GA. Tyto převodní algoritmy jsou formálně definovány v kapitole 6.8. V kapitole 6.9 jsou formálně definovány algoritmy pro simulaci aplikace testu a stanovení příkonových metrik. Algoritmy pro ohodnocení a výběr jedinců z populace jsou formálně definovány v kapitole 6.10. Způsob sestavení prvotní populace GA je vysvětlen v kapitole 6.11. Kapitola 6 a následující kapitoly jsou přínosem autora do uvedené problematiky.

6.1 Kompatibilita testu

Jedním z cílů při návrhu metodiky bylo dosažení kompatibility s profesionálními DfT nástroji. Aby bylo možné využít výstupy profesionálních generátorů testovacích vektorů jako vstup metodiky, je nutné definovat pravidla, která musí vygenerované testovací vektory splňovat. Různé generátory testovacích vektorů totiž interpretují aplikaci testu rozdílným

Tabulka 6.1: Příklad aplikace testu, strategie ALAP

SE	Synchronizační pulz	Testovací vektor	1. řetězec scan			2. řetězec scan	
			1. reg	2. reg	3. reg	1. reg	2. reg
1	t_1	–	$I_{1,1,3}$	–	–	X	–
1	t_2	–	$I_{1,1,2}$	$I_{1,1,3}$	–	$I_{1,2,2}$	X
1	t_3	–	$I_{1,1,1}$	$I_{1,1,2}$	$I_{1,1,3}$	$I_{1,2,1}$	$I_{1,2,2}$
0	t_4	V_1	$O_{1,1,1}$	$O_{1,1,2}$	$O_{1,1,3}$	$O_{1,2,1}$	$O_{1,2,2}$
1	t_5	V_1	$I_{2,1,3}$	$O_{1,1,1}$	$O_{1,1,2}$	X	$O_{1,2,1}$
1	t_6	V_1	$I_{2,1,2}$	$I_{2,1,3}$	$O_{1,1,1}$	$I_{2,2,2}$	X
1	t_7	V_1	$I_{2,1,1}$	$I_{2,1,2}$	$I_{2,1,3}$	$I_{2,2,1}$	$I_{2,2,2}$
0	t_8	V_2	$O_{2,1,1}$	$O_{2,1,2}$	$O_{2,1,3}$	$O_{2,2,1}$	$O_{2,2,2}$
1	t_9	V_2	X	$O_{2,1,1}$	$O_{2,1,2}$	X	$O_{2,2,1}$
1	t_{10}	V_2	X	X	$O_{2,1,1}$	X	X
1	t_{11}	V_2	X	X	X	X	X

způsobem, který může být odlišný od způsobu aplikace testu, která byla uvažována při návrhu metodiky. Z tohoto důvodu bude test splňující dále definovaná pravidla označován pojmem *kompatibilní test*. Test nesplňující tato pravidla bude označován pojmem *nekompatibilní test* (vzhledem k navržené metodice).

V tabulce 6.1 je uveden příklad aplikace kompatibilního testu na obvod obsahující dva řetězce scan. Je uvažována strategie testu ALAP (viz kapitola 4.3). Jsou uvažovány jednobitové vodiče, takže v každém scan registru může být uložena hodnota logická „0“ nebo logická „1“. Tento předpoklad neomezuje použití metodiky, protože obvody obsahující vícebitové vodiče, lze algoritmicky převést na funkčně identické obvody sestávající pouze z jednobitových vodičů. V příkladu v tabulce 6.1 první scan řetězec sestává ze tří scan registrů, druhý scan řetězec sestává ze dvou scan registrů. V příkladu jsou aplikovány dva testovací vektory V_1, V_2. Lze tedy rozlišit tři cykly testu. Počet cyklů testu je vždy o jedna vyšší než počet testovacích vektorů z důvodu nutnosti vyprázdnění scan řetězce na konci testu. Jednotlivé cykly testu jsou v tabulce 6.1 odděleny horizontální čarou. První dva testové cykly sestávají ze čtyř synchronizačních pulzů. Počet synchronizačních pulzů je u těchto cyklů vždy o jedna vyšší než počet registrů v nejdelším řetězci scan. V posledním testovém cyklu není aplikován testovací vektor, takže sestává jen ze tří synchronizačních pulzů. Počet synchronizačních pulzů u posledního cyklu je tedy v těchto případech vždy roven počtu registrů v nejdelším scan řetězci. V optimálním případě všechny scan řetězce obsahují shodný počet registrů, což však nelze u reálných obvodů vždy zajistit. Z tohoto důvodu je v tabulce 6.1 demonstrován případ scan řetězců s neshodným počtem scan registrů.

V příkladu v tabulce 6.1 je uvažována realizace scan registru využívající paměťového prvku a multiplexoru (viz obr. 2.6). Mezi sériovým a paralelním načítáním/vyčítáním dat je voleno pomocí primárního vstupu SE. Logické hodnoty přiváděné na primární vstup SE jsou zapsány v prvním sloupci tabulky 6.1. Sériový posuv scan řetězcem je zvolen v případě $SE = 1$, v případě $SE = 0$ jsou načtená data ze scan registrů paralelně aplikována na testovaný obvod a zároveň dochází k paralelnímu načtení odezvy obvodu do scan registrů. V případě běžného funkčního režimu je též $SE = 0$ a scan registry fungují jako běžné registry (viz kapitola 2.6). Aktuální hodnota SE je uvedena v prvním sloupci tabulky 6.1. Druhý sloupec definuje jednotlivé synchronizační pulzy. Pro zjednodušení jsou uvažo-

vány registry reagující na náběžnou hranu synchronizačního signálu, kdy zápis do registru je realizován jedním synchronizačním pulzem. Třetí sloupec uvádí posloupnost aplikace testovacích vektorů a definuje okamžiky, ve kterých se předpokládá nastavení testovacích vektorů na primární vstupy (toto rozlišení je podstatné z hlediska stanovení příkonových metrik). Čtvrtý sloupec definuje první řetězec scan. Pátý sloupec definuje druhý řetězec scan. V těchto dvou posledních sloupcích jsou definovány obsahy jednotlivých scan registrů pro jednotlivé časové okamžiky.

Každý řádek tabulky 6.1 reprezentuje situaci po příchodu daného synchronizačního pulzu. První řádek tabulky 6.1 reprezentuje situaci po příchodu synchronizačního pulzu t_1. Primární vstupy jsou nastaveny na uživatelem předem definovanou inicializační hodnotu INIT_V (logická „0“ nebo logická „1“), což je v tabulce reprezentováno symbolem „−“. Do prvního registru prvního scan řetězce je načtena hodnota $I_{1,1,3}$, jejíž označení lze dekódovat jako vstupní hodnota prvního scan vektoru (**1**, 1, 3), prvního scan řetězce (1, **1**, 3), určená pro třetí scan registr (1, 1, **3**).

Druhý scan řetězec je kratší o jeden registr, takže do něj musí být načtena tzv. vycpávka, která je v tabulce označená symbolem „X“. Jako vycpávka může být teoreticky použita libovolná hodnota. Pro minimalizaci počtu překlopení během posuvu hodnoty scan řetězcem, je nejvhodnější jako vycpávku použít hodnotu určenou pro poslední scan registr, tedy v tomto případě by to byla hodnota $I_{1,2,2}$.

Po příchodu synchronizačního pulzu t_2 (druhý řádek tabulky) dochází k posuvu hodnot scan řetězci. Tedy hodnota $I_{1,1,3}$ je přesunuta do druhého scan registru prvního scan řetězce. Do prvního registru je nyní načtena hodnota $I_{1,1,2}$. Ve druhém scan řetězci je vycpávka přesunuta do druhého scan registru a do prvního scan registru je načtena hodnota $I_{1,2,2}$.

Po příchodu synchronizačního pulzu t_3 (třetí řádek tabulky) dochází k dalšímu posuvu hodnot scan řetězci. Po tomto posuvu se všechny hodnoty nacházejí ve správných scan registrech. Tedy v prvním scan řetězci bude v prvním scan registru uložena hodnota $I_{1,1,1}$, ve druhém scan registru hodnota $I_{1,1,2}$, ve třetím scan registru hodnota $I_{1,1,3}$. Ve druhém scan řetězci bude v prvním scan registru uložena hodnota $I_{1,2,1}$ a ve druhém scan registru hodnota $I_{1,2,2}$. Protože se jedná o strategii testu ALAP, tak nyní může dojít k nastavení testovacího vektoru V_1 na primární vstupy obvodu.

Po příchodu synchronizačního pulzu t_5 je sejmuta odezva obvodu na primárních výstupech (není v tabulce vyznačeno) a taktéž je sejmuta odezva obvodu pomocí jednotlivých scan registrů. V prvním scan řetězci budou nyní uloženy odezvy obvodu $O_{1,1,1}$, $O_{1,1,2}$, $O_{1,1,3}$. Ve druhém scan řetězci budou uloženy odezvy obvodu $O_{1,2,1}$, $O_{1,2,2}$. Tímto je první cyklus testu ukončen.

Ve druhém cyklu testu sestávajícím ze synchronizačních pulzů t_5 až t_8, je odezva obvodu sériově vyčtena ze scan řetězců a zároveň je načten další scan vektor do prvního a druhého řetězce scan. Tedy po příchodu synchronizačního pulzu t_7 (sedmý řádek tabulky) budou v prvním scan řetězci načteny hodnoty $I_{2,1,1}$, $I_{2,1,2}$, $I_{2,1,3}$ a ve druhém scan řetězci budou načteny hodnoty $I_{2,2,1}$, $I_{2,2,2}$. Na primární vstupy obvodu tedy může být nastaven testovací vektor V_2. Po příchodu synchronizačního pulzu t_8 (osmý řádek tabulky), budou v prvním scan řetězci zachyceny odezvy obvodu $O_{2,1,1}$, $O_{2,1,2}$, $O_{2,1,3}$ a ve druhém scan řetězci budou zachyceny odezvy $O_{2,2,1}$, $O_{2,2,2}$.

Ve třetím cyklu testu sestávajícím ze synchronizačních pulzů t_9 až t_{11} jsou ze scan řetězců sériově vyčteny odezvy obvodu získané v předchozím cyklu testu, čímž je test ukončen.

Z výše uvedeného příkladu lze tedy pravidla pro kompatibilní test shrnout následovně:

- Test musí být možné rozdělit na testové cykly během kterých jsou aplikovány jednot-

Tabulka 6.2: Příklad zjednodušeného zápisu aplikace testu

Testový cyklus	Testovací vektor	Fáze scan řetězce			Odezva obvodu
1	V_1	i_1	$o_{X,1}$	a_1	R_1
2	V_2	i_2	o_1	a_2	R_2
3	V_3	i_3	o_2	a_3	R_3
4	X	$i_{X,1}$	o_3		X

livé testovací vektory nebo dochází k vyprázdnění řetězce scan.

- Počet testových cyklů musí být u obvodů s plným scanem o jedna vyšší, než je počet testovacích vektorů. U obvodů bez řetězce scan musí být roven počtu testovacích vektorů.
- V případě plného řetězce scan testový cyklus sestává z jednotlivých synchronizačních pulzů, jejichž počet je o jedna vyšší než počet registrů v nejdelším řetězci scan.
- V každém testovém cyklu (kromě posledního v případě plného řetězce scan) je aplikován odpovídající testovací vektor.
- V případě plného řetězce scan je v každém testovém cyklu kromě posledního do každého scan řetězce sériově načten odpovídající scan vektor. V posledním testovém cyklu může být sériově načítána libovolná hodnota a dochází k sériovému vyprázdnění scan řetězce.
- V případě plného řetězce scan je v každém testovém cyklu sériově vyčtena předchozí odezva obvodu.
- Do všech scan řetězců, které jsou kratší než nejdelší scan řetězec, musí být v každém testovém cyklu načtena odpovídající vycpávka.

6.2 Úprava na kompatibilní test

Některé generátory testovacích vektorů generují nekompatibilní testy, takže je nutné je před aplikací metodiky upravit na kompatibilní tvar. Pro diskusi nejrozšířenějších případů budeme uvažovat obvod s jedním řetězcem scan. Princip je snadno rozšiřitelný i pro obvody s více řetězci scan. Pro zjednodušení použijeme schématický zápis aplikace testu, viz tabulka 6.2. Dále využijeme možnosti rozdělit každý testový cyklus na logicky samostatné celky, které budeme nazývat *fáze*. Tyto fáze budou označovány v_x, r_x, i_x, a_x, o_x, kde x je přirozené číslo určující pořadí testového cyklu, ke kterému fáze náleží. V tomto okamžiku je třeba zmínit, že se jedná pouze o logické dělení a jednotlivé fáze nejsou vzájemně časově disjunktní. Jednotlivé fáze v, r, i, a, o mají následující význam. Ve fázi v_x dochází k aplikaci testovacího vektoru V_x přes primární vstupy obvodu. Ve fázi r_x dochází k sejmutí odezvy obvodu R_x přes primární výstupy. Během fáze i_x je provedeno sériové načtení scan vektoru I_x do scan řetězců. Ve fázi a_x dochází k aplikaci scan vektoru a paralelní načtení odezvy obvodu do scan registrů a nakonec během fáze o_x je provedeno sériové vyčtení odezvy obvodu O_x.

V tabulce 6.2 je zjednodušeně zapsán test sestávající ze čtyř testových cyklů. První cyklus testu je vyznačen na prvním řádku tabulky. V tomto cyklu je na primární vstupy

Tabulka 6.3: Převod nekompatibilního testu typu 1 na kompatibilní test

Test. cykl.	Test. vekt.	Fáze scan řetězce			Odez. CUA		Test. cykl.	Test. vekt.	Fáze scan řetězce			Odez. CUA
1	V_1	i_1	$o_{X,1}$		R_1		1	V_1	i_1	$o_{X,1}$	a_1	R_1
	V_2				R_2		2	V_2	i_1	$o_{X,2}$	a_1	R_2
	V_3	a_1			R_3	$\Rightarrow$	3	V_3	i_1	$o_{X,3}$	a_1	R_3
2	V_4	i_2	o_1	a_2	R_4		4	V_4	i_2	o_1	a_2	R_4
3	X	$i_{X,1}$	o_2		X		5	X	$i_{X,1}$	o_2		X

obvodu nastaven testovací vektor V_1, do scan řetězce je sériově načten scan vektor I_1, zároveň je ze scan řetězce sériově vyčtena odezva $O_{X,1}$. Index $X, 1$ označuje, že se jedná o první odezvu, která není podstatná pro výsledek testu (jedná se totiž o vektor tvořený náhodnými hodnotami, na které byly inicializovány jednotlivé scan registry před zahájením testu). Dále dochází k aplikaci testovacího vektoru V_1 a scan vektoru I_1 a k paralelnímu načtení odezvy obvodu do scan řetězce (fáze a_1) a na primárních výstupech obvodu je sejmuta odezva R_1. Tímto je první cyklus dokončen.

Ve druhém cyklu testu (druhý řádek tabulky) je na primární vstupy obvodu nastaven testovací vektor V_2, do scan řetězce je sériově načten scan vektor I_2, zároveň je ze scan řetězce sériově vyčtena odezva obvodu O_1 na předchozí testovací vektory. Dále dochází k aplikaci testovacího vektoru V_2 a scan vektoru I_2 a k paralelnímu načtení odezvy obvodu do scan řetězce (fáze a_2). Na primárních výstupech obvodu je sejmuta odezva R_2. Obdobně pro třetí testový cyklus. Ve čtvrtém testovém cyklu je sériově vyčtena odezva obvodu O_3 na předchozí testovací vektory. Při sériovém vyčítání O_3 je zároveň sériově načten scan vektor $I_{X,1}$, na jehož hodnotě však z hlediska testu nezáleží. Tento scan vektor slouží pouze pro vyprázdnění scan řetězce. Tímto je test ukončen.

Pomocí principu představeného v tabulce 6.2 budeme dále definovat nejčastěji se vyskytující typy nekompatibilních testů a způsob jejich převedení na kompatibilní testy. U komerčních generátorů testovacích vektorů se nejčastěji vyskytují dva typy nekompatibilních testů, které lze oba převést na kompatibilní testy. Tyto dva typy lze neformálně popsat následujícím způsobem:

1. V jednom testovém cyklu dochází k aplikování více než jednoho testovacího vektoru přes primární vstupy obvodu. Tento případ bude dále označován jako nekompatibilní test typu 1 a je demonstrován v levé části tabulky 6.3. Odpovídající kompatibilní test je uveden v pravé části tabulky 6.3.

2. V jednom testovém cyklu dochází k načtení více než jednoho scan vektoru. Tento případ bude dále označován jako nekompatibilní test typu 2 a je demonstrován v levé části tabulky 6.4. Odpovídající kompatibilní test je uveden v pravé části tabulky 6.4.

V levé části tabulky 6.3 je uveden příklad nekompatibilního testu typu 1. V prvním cyklu testu jsou totiž aplikovány tři testovací vektory V_1, V_2, V_3. V pravé části tabulky 6.3 je ten stejný test po převodu na kompatibilní formát. Převod spočívá v přidání dvou testových cyklů, ve kterých budou aplikovány testovací vektory V_2, V_3. Do těchto dvou nových testových cyklů jsou přidány scan fáze i_1, a_1, které jsou replikovány z prvního řádku tabulky. Sériově vyčítané odezvy obvodu $O_{X,2}$, $O_{X,3}$ nejsou při aplikaci testu použity.

Tabulka 6.4: Převod nekompatibilního testu typu 2 na kompatibilní test

Test. cykl.	Test. vekt.	Fáze scan řetězce			Odez. CUA		Test. cykl.	Test. vekt.	Fáze scan řetězce			Odez. CUA
1	V_1	i_1	o_X	a_1	R_1		1	V_1	i_1	o_X	a_1	R_1
		i_2	o_1	a_2	R_2	$\Rightarrow$	2	V_1	i_2	o_1	a_2	R_2
2	V_2	i_3	o_2	a_3	R_3		3	V_2	i_3	o_2	a_3	R_3
3	X	i_X	o_3		X		4	X	i_X	o_3		X

V levé části tabulky 6.4 je uveden příklad nekompatibilního testu typu 2. V prvním cyklu testu jsou totiž aplikovány dva scan vektory I_1, I_2. V pravé části tabulky 6.3 je ten stejný test po převodu na kompatibilní formát. Převod spočívá v přidání jednoho testového cyklu, ve kterém je znovu aplikován testový vektor V_1. V praxi je tento převod spíše jen formální záležitostí spočívající ve zvýšení počtu testových cyklů.

Uvedené způsoby lze vzájemně kombinovat, takže je možné převádět i mnohem složitější případy. Při dodržení výše stanovené logiky aplikace testu lze nalézt způsoby konverze i pro další výše nedefinované nekompatibilní testy.

6.3 Genetický algoritmus

Metodika využívá GA pro prohledávání stavového prostoru. GA se používají k prohledávání rozsáhlých stavových prostorů nejrůznějších úloh [50]. Nejčastěji se uplatňují u vyhledávacích a optimalizačních problémů s NP složitostí. GA se těší značné popularitě díky schopnosti nacházet kvalitní řešení složitých problémů v přijatelném čase. Nevýhodou GA je, že nejsou aplikovatelné na všechny problémy a že nemusí nalézt nejkvalitnější řešení.

GA jsou třídou evolučních algoritmů. Pro svoji funkci používají techniky inspirované evoluční biologií jako je dědičnost, mutace, přirozený výběr a křížení. Koncepce GA vznikla v roce 1973 ze studií celulárních automatů prováděné na Michiganské univerzitě. Za autora GA je považován John Holland. GA se typicky implementují jako počítačová simulace, ve které se populace kandidátních řešení postupně vyvíjí dle evolučních pravidel, čímž jsou postupně nacházena stále kvalitnější řešení. Chromozóm představuje abstraktní reprezentaci řešeného problému. Chromozóm je tvořen jednotlivými geny. Jednotlivá kandidátní řešení (daná možnými kombinacemi nul a jedniček v chromozómu) se nazývají individua. Evoluce začíná od prvotní populace, která je často vygenerována pomocí nějakého známého algoritmu anebo se náhodně zvolí. Algoritmus pak postupuje v iteracích po jednotlivých generacích. V každé generaci je stanovena kvalita všech řešení obsažených v populaci. Kvalita řešení je ohodnocena pomocí funkce fitness, která každému jedinci přiřadí číselnou hodnotu, jež odpovídá kvalitě daného řešení. Dále je proveden výběr jedinců dle jejich hodnoty fitness a vybraní jedinci jsou modifikováni (aplikací operátorů mutace) a rekombinovány (aplikací operátorů křížení), čímž vzniká nová generace jedinců. Sémantika a pravděpodobnost jednotlivých operátorů jsou parametry, na jejichž vhodné volbě často závisí kvalita nalezených řešení. Jednotlivé iterace algoritmu pokračují tak dlouho, dokud není nalezeno řešení požadované kvality nebo neproběhne předem definovaný počet iterací. Genetický algoritmus lze formálně definovat pomocí následující definice:

6.3.1 Formální definice GA

Definice 6.3.1.

- *Nechť S je množina všech možných řešení zkoumané úlohy a*
- *nechť N je množina individuí, pro kterou platí $N = B^l$, kde $B = \{b_1, b_2, \ldots, b_n\}$ představuje alelu, jež může nabývat hodnot b_1, b_2, …, b_n a l určuje délku chromozómu a N je množina všech řetězců nad B,*
- *nechť existuje relace $\Delta : N \to S$ přiřazující každému chromozómu řešení z prostoru řešení S,*
- *nechť existuje konstanta $\lambda : \lambda \in \aleph$ určující velikost populace (počet jedinců v každé generaci),*
- *nechť existuje relace $\Phi : (N \to S) \to \Re^+$, přiřazující kladná reálná čísla jedincům dle jejich kvality, tzv. fitness funkce,*
- *nechť existuje množina $\Omega = (\omega_{\Theta 1}, \omega_{\Theta 2}, \ldots, \omega_{\Theta k} | \omega_{\Theta i} : N^\lambda \to C^\lambda)$, definující genetické operátory $\omega_{\Theta i}$. Nechť ke každému operátoru existuje odpovídající pravděpodobnost $p_i : p_i \in \Re \wedge p_i \in \langle 0; 1 \rangle$ definující pravděpodobnost aplikace operátoru $\omega_{\Theta i}$,*
- *nechť operátor $s : N^\lambda \to N^\lambda$ je výběrový operátor, který slouží pro výběr jedinců z populace dle jejich kvality (hodnoty fitness),*
- *nechť relace $\tau : N^\lambda \to \{true, false\}$ definuje ukončující podmínku (algoritmus bude ukončen v případě dosažení hodnoty true),*
- *nechť relace $\Xi : N^\lambda \to N^\lambda$ definuje přechodovou funkci mezi jednotlivými generacemi,*

potom 7-ce $GA = (N, \Delta, \Phi, \Omega, \Xi, \lambda, \tau)$ představuje základní genetický algoritmus.

Definovaná metodika využívá základní genetický algoritmus a základní genetické operátory. Mezi základní genetické operátory lze počítat operátor mutace a operátor křížení. Tyto operátory budou definovány v následujících kapitolách.

6.3.2 Operátor mutace

Mutace je dědičná změna genotypu. Operátor mutace se zpravidla používá pro omezení konvergence algoritmu do lokálního minima, případně umožňuje nalézat nová řešení díky náhodným změnám genotypu.

Definice 6.3.2. *Nechť existuje operátor $m_{p_m} : N \to N \wedge m_{p_m} \in \Omega$, modifikující genotyp jedince náhodným způsobem a nechť pravděpodobnost aplikace operátoru je určena hodnotou p_m, potom m_{p_m} buď nazýván operátor mutace.*

Nejčastěji se používá tzv. poziční mutace, kdy se stochasticky vybere gen, u kterého proběhne mutace. Příklad aplikace operátoru poziční mutace lze vidět v tabulce 6.5

V prvním řádku tabulky 6.5 je uveden příklad chromozómu před aplikací operátoru mutace, ve druhém řádku je ten stejný chromozóm po aplikaci operátoru mutace v případě, že byl zvolen 3. gen pro mutaci. Kódová sekvence b_{i_3} byla tedy mutována na b'_{i_3}.

Tabulka 6.5: Příklad aplikace operátoru poziční mutace

Původní chromozóm	b_{i_1}	b_{i_2}	b_{i_3}	...	b_{i_l}
Nový chromozóm	b_{i_1}	b_{i_2}	b'_{i_3}	...	b_{i_l}

Tabulka 6.6: Příklad aplikace jednobodového operátoru křížení pro osu křížení $o = 2$

	Chromozóm 1					Chromozóm 2				
Rodiče	b_{i_1}	b_{i_2}	b_{i_3}	...	b_{i_l}	b_{j_1}	b_{j_2}	b_{j_3}	...	b_{j_l}
Potomci	b_{i_1}	b_{i_2}	b_{j_3}	...	b_{j_l}	b_{j_1}	b_{j_2}	b_{i_3}	...	b_{i_l}

6.3.3 Operátor křížení

Křížení je způsob, jak z několika chromozomů vytvořit nový chromozom. Křížení se účastní nejčastěji dva chromozomy (tzv. rodiče), ze kterých po aplikaci operátoru křížení vznikne nový chromozom (tzv. potomek) dědící vlastnosti rodičů. Často se používá operátor křížení vytvářející dva potomky z důvodu zachování počtu jedinců v populaci. Existuje více způsobu křížení, nejběžnějším je tzv. jednobodový operátor křížení. V tomto případě se zvolí tzv. osa křížení o, kolem které jsou rodiče „překříženi“. Jako osa křížení se většinou volí prostřední gen chromozomu. Příklad aplikace operátoru křížení lze vidět v tabulce 6.6.

Definice 6.3.3. *Nechť existuje operátor* $c_{p_c} : (N \times N) \rightarrow (N \times N) \wedge m_{p_m} \in \Omega$*, který provádí kombinaci dvou jedinců a nechť pravděpodobnost aplikace operátoru je určena hodnotou* p_c*, potom* c_{p_c} *buď nazýván operátor křížení.*

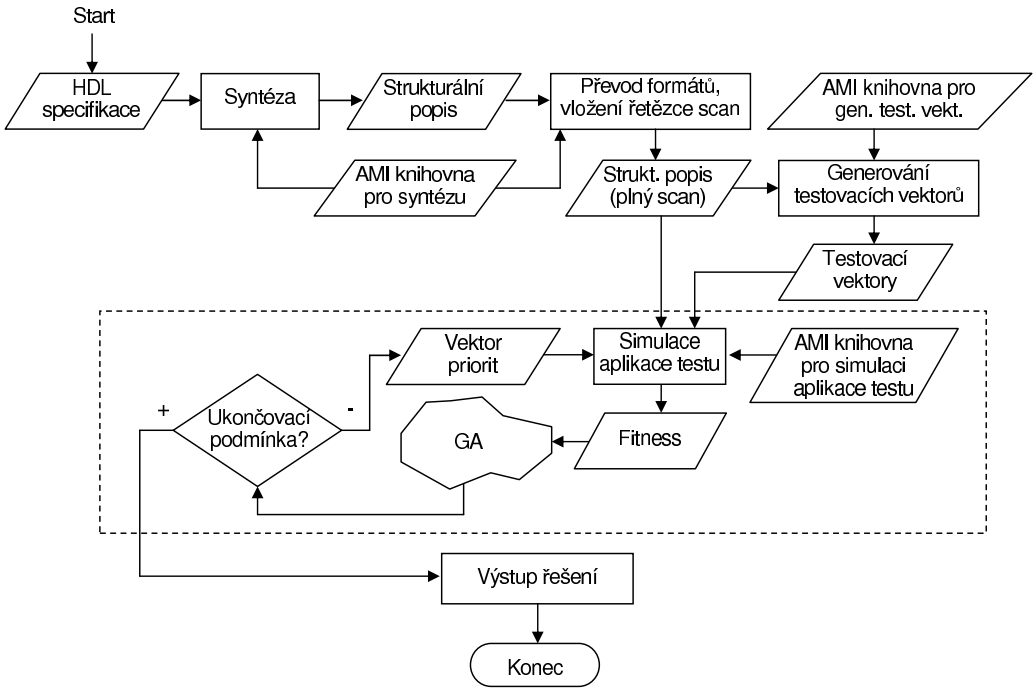
V prvním řádku tabulky 6.6 je uveden příklad dvou rodičů, kteří jsou označeni jako „Chromozóm 1“ a „Chromozóm 2“. Ve druhém řádku jsou pak uvedeni potomci vzniklí aplikací operátoru kříženi kolem osy $o = 2$. První potomek vznikne zkopírováním kódové sekvence prvního rodiče nalevo od osy křížení a doplněním kódové sekvence druhého rodiče napravo od osy křížení. Druhý potomek je vytvořen symetricky, tedy kódová sekvence nalevo od osy křížení je zkopírována z druhého rodiče a kódová sekvence napravo od osy křížení je doplněna z prvního rodiče.

6.3.4 Elitismus

Elitismus umožňuje zachování nejkvalitnějších řešení při přechodu mezi jednotlivými generacemi. V případě n - prvkového elitismu přechází n nejkvalitnějších jedinců z aktuální generace automaticky do následující generace. V případě použití elitismu je zajištěno, že výstupem GA nebude řešení horší, než řešení obsažené v prvotní populaci.

6.4 Princip metodiky

Zjednodušené blokové schéma navržené metodiky lze vidět na obr. 6.1. Na obr. 6.1 kosodélníky představují jednotlivá data a obdélníky způsob jejich zpracování. Vstupem metodiky je HDL specifikace CUA, která je namapována na definovanou technologickou knihovnu. Do získaného strukturálního popisu je vložen plný řetězec (řetězce) scan a jsou vygenerovány testovací vektory. Tím je ukončena přípravná fáze. Vlastní optimalizační metoda je na obr. 6.1 ohraničena obdélníkem zakresleným čárkovanou čarou. Srdcem metody je GA.



Obrázek 6.1: Blokové schéma metodiky

Během inicializace je vytvořena prvotní populace jedinců. Dokud není splněna ukončovací podmínka (přednastavený počet iterací), probíhá vývoj populace po jednotlivých generacích. V každém kroku dochází k ohodnoceni všech jedinců populace. Ohodnocení spočívá v dekódování genomu jedince na vektor priorit, jež určuje pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce (řetězců) scan. Tyto vektory priorit jsou využity během simulace aplikace testu. Simulace je prováděna nad technologickou knihovnou. Výsledkem simulace je hodnota odpovídající příkonu obvodu během aplikace testu ve zvolené metrice (např. NTC). Tato hodnota je přepočtena na hodnotu fitness, jež definuje kvalitu řešení. Následně dochází k výběru, křížení a mutaci jedinců populace pomocí genetických operátorů, čímž vzniká nová generace jedinců. U GA se počítá s elitismem, takže nedochází ke ztrátě nejkvalitnějších řešení nalezených během vývoje populace. Jakmile je dosaženo ukončovací podmínky, je z populace vybrán nejkvalitnější jedinec, jehož genom je dekódován a je stanoveno pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce (řetězců scan). Tyto informace jsou výstupem metodiky.

6.5 Formální model pro definování principů metodiky

Pro řešení problémů souvisejících s metodikou prezentovanou v této práci bylo využito formálního modelu. Model využívá aparátu z teorie množin. Nad tímto modelem jsou dále vystavěny jednotlivé algoritmy pro řešení problému. Jako základ pro formální model bylo použito části modelu definovaného v [55]. Tento model byl dále rozšířen o nové prvky. Vytvořený model reflektuje strukturální, diagnostické i elektrické vlastnosti CUA. Ze strukturálního pohledu model popisuje primární rozhraní CUA, jednotlivé obvodové prvky, ze kterých je CUA složen, rozhraní těchto prvků (tzv. brány), spoje propojující jednotlivá roz-

hraní. Z diagnostického pohledu model definuje topologii řetězců scan, jednotlivé testovací vektory a pořadí jejich aplikace. Model definuje i přechodové funkce jednotlivých obvodových prvků a příkon obvodových prvků v jednotlivých příkonových metrikách (viz 3.1.4) při přechodech mezi jednotlivými stavy.

6.5.1 Množina bvodových prvků E

Definice 6.5.1. *Nechť R je množina všech registrů v CUA, $SC \subseteq R$ je množina všech scan registrů v CUA, MX je množina všech multiplexorů v CUA, FU je množina všech funkčních prvků v CUA, potom $E = R \cup SR \cup MX \cup FU$ je množina všech obvodových prvků.*

Množina E definuje všechny obvodové prvky CUA.

Příklad 6.5.1:
$R = \{R_1, R_2\}$
$SC = \{R_1, R_2\}$
$MX = \{MX_1\}$
$FU = \{ADD_1, SUB_1\}$
$E = \{R_1, R_2, MX_1, ADD_1, SUB_1\}$

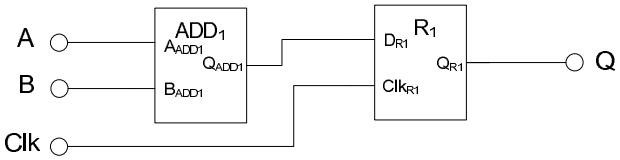
V příkladu 6.5.1 se jedná o obvod, který obsahuje dva registry R_1, R_2, dále jeden multiplexor MX_1 a dva funkční prvky – sčítačku ADD_1 a odčítačku SUB_1. Jedná se o obvod s plným scanem, protože $R = SC$. Pokud by bylo $SC \subset R$ jednalo by se o obvod s částečným scanem – tento případ však není metodikou podporován. Pokud by bylo $SC = R = \emptyset$, jednalo by se o kombinační obvod. Pokud by bylo $R \neq \emptyset \wedge SC = \emptyset$, jednalo by se o sekvenční obvod bez řetězce scan – tento případ také není metodikou podporován. V případě, že je počet řetězců scan větší než jedna, musí být všechny scan registry ze všech scan řetězců obsaženy v SC.

6.5.2 Množina obvodových bran P

Definice 6.5.2. *Nechť IN je množina všech datových vstupních bran (kromě primárních), OUT množina všech datových výstupních bran (kromě primárních) a CI množina všech synchronizačních a řídicích bran, potom množina $P = IN \cup OUT \cup CI$ je množina všech bran všech obvodových prvků.*

Množina P definuje všechna rozhraní (brány) všech obvodových prvků. V modelu jsou uvažovány jednobitové brány, protože takové brány se vyskytují u prvků definovaných v knihovně AMI. Z těchto prvků je však možné snadno sestavit prvky s vícebitovými branami. V příkladu 6.5.2 je uvedena množina P pro obvod z obrázku 6.2 obsahující dva prvky sčítačku ADD_1 (bez přenosu) a registr $R1_1$.

Příklad 6.5.2:
$R = \{R_1\}$
$FU = \{ADD_1\}$
$E = \{R_1, ADD_1\}$
$IN = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}\}$
$OUT = \{Q_{R_1}, Q_{ADD_1}\}$
$CI = \{Clk_{R_1}\}$



Obrázek 6.2: Příklad jednoduchého obvodu

$P = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}, Q_{R_1}, Q_{ADD_1}, Clk_{R_1}\}$

V příkladu 6.5.2 sčítačka ADD_1 má vstupní datové brány A_{ADD_1}, B_{ADD_1} a datovou výstupní bránu Q_{ADD_1}. Registr R_1 má datovou vstupní bránu D_{R_1}, datovou výstupní bránu Q_{R_1} a řídicí bránu Clk_{R_1} pro vstup synchronizačního signálu.

6.5.3 Relace *in*

Definice 6.5.3. *Nechť existuje relace* $in : E \rightarrow 2^{IN}$*, která splňuje následující pravidla:*

1. $\forall e \in E : in(e) = \{p | p \in IN \wedge p$ *je brána prvku* $e\}$,
2. $\forall e_1, e_2 \in E : e_1 \neq e_2 \Leftrightarrow in(e_1) \cap in(e_2) = \emptyset$,

potom relace in přiřazuje vstupní brány jednotlivým obvodovým prvkům.

První pravidlo definuje, že se musí jednat o vstupní datovou bránu prvku. Druhé pravidlo definuje, že prvky nemohou mezi sebou sdílet brány.

Příklad 6.5.3:
$E = \{R_1, ADD_1\}$
$IN = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}\}$

$in(R_1) = \{D_{R_1}\}$
$in(ADD_1) = \{A_{ADD_1}, B_{ADD_1}\}$

V příkladu 6.5.3 je uvažován stejný obvod jako v příkladu 6.5.2. Pro zjednodušení nejsou uváděny všechny množiny modelu, ale pouze množiny relevantní pro tento příklad (tento přístup bude používán i u dalších příkladů). V tomto případě tedy relace $in = \{(R_1, \{D_{R_1}\}), (ADD_1, \{A_{ADD_1}, B_{ADD_1}\})\}$, takže např. $in(r_1) = \{D_{R_1}\}$ značí, že vstupní branou registru R_1 je D_{R_1}. Podobně $in(ADD_1) = \{A_{ADD_1}, B_{ADD_1}\}$ značí, že vstupními branami sčítačky ADD_1 jsou brány vA_{ADD_1}, B_{ADD_1}.

6.5.4 Relace *out*

Definice 6.5.4. *Nechť existuje relace* $out : E \rightarrow 2^{OUT}$*, která splňuje následující pravidla:*

1. $\forall e \in E : out(e) = \{p | p \in OUT \wedge p$ *je brána prvku* $e\}$,
2. $\forall e_1, e_2 \in E : e_1 \neq e_2 \Leftrightarrow out(e_1) \cap out(e_2) = \emptyset$,

potom relace out přiřazuje výstupní brány jednotlivým obvodovým prvkům.

První pravidlo definuje, že se musí jednat o výstupní datovou bránu prvku. Druhé pravidlo definuje, že prvky nemohou mezi sebou sdílet brány.

Příklad 6.5.4:
$E = \{R_1, ADD_1\}$
$OUT = \{Q_{R_1}, Q_{ADD_1}\}$

$out(R_1) = \{Q_{R_1}\}$
$out(ADD_1) = \{Q_{ADD_1}\}$

V příkladu 6.5.4 je uvažován stejný obvod jako v příkladu 6.5.2. V tomto případě tedy relace $out = \{(R_1, \{Q_{R_1}\}), (ADD_1, \{Q_{ADD_1}\})\}$, tedy $out(R_1) = \{Q_{R_1}\}$ značí, že výstupní branou registru R_1 je Q_{R_1}. Podobně $out(ADD_1) = \{Q_{ADD_1}\}$ značí, že výstupní branou sčítačky ADD_1 je brána Q_{ADD_1}.

6.5.5 Relace C

Definice 6.5.5. *Nechť PI je množina všech primárních vstupních bran obvodu, PO množina všech primárních výstupních bran obvodu a P množina všech ostatních bran v obvodu a nechť existuje relace $C \subseteq PI \cup PO \cup P) \times (PI \cup PO \cup P)$:*

1. $\forall p_1, p_2 \in P : C = \{(p_1, p_2) |$ *v obvodu existuje spoj mezi branami* $p_1, p_2\}$,

potom relace C definuje všechny metalické spoje v obvodu.

Relace C je tranzitivní, reflexivní a symetrická, což je zjevné z vlastností metalického spoje.

Příklad 6.5.5:
$E = \{R_1, ADD_1\}$
$P = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}, Q_{R_1}, Q_{ADD_1}, Clk_{R_1}\}$
$PI = \{A, B, Clk\}$
$PO = \{Q\}$

$C = \{(A, A), (B, B), (Q, Q), (Clk, Clk), (D_{R_1}, D_{R_1}), (A_{ADD_1}, A_{ADD_1}),$
$(B_{ADD_1}, B_{ADD_1}), (Q_{R_1}, Q_{R_1}), (Q_{ADD_1}, Q_{ADD_1}), (Clk_{R_1}, Clk_{R_1}), (A, A_{ADD_1}), (A_{ADD_1}, A),$
$(B, B_{ADD_1}), (B_{ADD_1}, B), (Clk, Clk_{R_1}), (Clk_{R_1}, Clk), (Q_{ADD_1}, D_{R_1}), (D_{R_1}, Q_{ADD_1}),$
$(Q_{R_1}, Q), (Q, Q_{R_1})\}$

V příkladu 6.5.4 je uvažován stejný obvod jako v příkladu 6.5.2. Např. prvek (Q_{ADD_1}, D_{R_1}) relace C značí, že brána Q_{ADD_1} je propojena s branou D_{R_1}.

6.5.6 Obor hodnot signálu D

Definice 6.5.6. *Nechť symboly 0, 1, Z představují signály „logická nula“, „logická jedna“ a stav „vysoké impedance“ a nechť symbol $\updownarrow$ představuje impulz (změnu z 0 na 1 a pozdější návrat do 0) , potom množina $D = \{0, 1, Z, \updownarrow\}$ definuje obor hodnot, jež mohou nabývat jednotlivé obvodové brány během simulace aplikace testu.*

Během simulace aplikace testu mohou jednotlivé obvodové brány nabývat pouze hodnot z množiny D. Symbol $\updownarrow$ je použit pro zjednodušené modelování synchronizačních pulzů.

6.5.7 Příkonové metriky MT

Definice 6.5.7. *Nechť symboly M_i představují jednotlivé příkonové metriky ($i = 1 \dots n_m$, kde n_m je počet uvažovaných metrik), potom množina $MT = \{M_1, M_2, \dots, M_{n_m}\}$ představuje všechny uvažované příkonové metriky.*

Množina MT definuje všechny příkonové metriky, jež mohou být použity během simulace aplikace testu. Ve vytvořené implementaci jsou podporovány metriky NTC, $WNTC$, WSA, tedy $M_1 \equiv NTC$, $M_2 \equiv WNTC$, $M_3 \equiv WSA$.

6.5.8 Relace pin_ord

Definice 6.5.8. *Nechť existuje relace $pin\_ord : PI \rightarrow \aleph$, která splňuje následující pravidla:*

1. $\forall p_1, p_2 \in PI : p_1 \neq p_2 \rightarrow pin\_ord(p_1) \neq pin\_ord(p_2)$,
2. $\exists p_3 \in PI : pin\_ord(p_3) = 0$,
3. $\exists p_4 \in PI : pin\_ord(p_4) = |PI| - 1 \wedge \forall p_5 \in PI : pin\_ord(p_5) \leq pin\_ord(p_4)$,

potom relace pin_ord jednoznačně identifikuje každou primární vstupní bránu pomocí čísla z množiny přirozených čísel.

První pravidlo definuje, že každé primární vstupní bráně je přiřazeno unikátní číslo z množiny přirozených čísel (včetně 0). Druhé pravidlo definuje, že nejmenší přiřazené číslo je 0. Dle druhého pravidla totiž musí být 0 přiřazena. Množina přirozených čísel již žádné menší číslo než 0 neobsahuje, a proto musí být všechna ostatní přiřazená čísla vyšší než 0 (stejná být nemohou kvůli prvnímu pravidlu). Třetí pravidlo definuje, že nejvyšší přiřazené číslo je $|PI| - 1$. Relace je definována pro všechny prvky z množiny PI, tudíž z druhého a třetího pravidla vyplývá, že mezi čísly 0 až $|PI| - 1$ neexistuje číslo, které by nebylo přiřazeno některé primární vstupní bráně. Identifikační čísla bran budou využívána v simulačních algoritmech, jež budou definovány v kapitole 6.9.

Příklad 6.5.6:
$PI = \{A, B, Clk\}$

$pin\_ord = \{(A, 0), (B, 1), (Clk, 2)\}$

V příkladu 6.5.6 je uvažován stejný obvod jako v příkladu 6.5.2. Primárním vstupním branám budou dle definice relace pin_ord přiřazena identifikační čísla 0, 1, 2. Existuje tedy celkem $|PI|!$ možností přiřazení těchto tří čísel primárním vstupním branám, $3! = 6$. Všechny tyto způsoby přiřazení čísel branám jsou dle definice korektní. Konkrétně použitý způsob přiřazení není vzhledem k funkci dále definovaných algoritmů podstatný. Jedno z možných přiřazení lze vidět v příkladu 6.5.6

6.5.9 Relace in_ord

Definice 6.5.9. *Nechť existuje relace $in\_ord : IN \rightarrow \aleph$, která splňuje následující pravidla:*

1. $\forall p_1, p_2 \in IN : p_1 \neq p_2 \rightarrow in\_ord(p_1) \neq in\_ord(p_2)$,

2. $\exists p_3 \in IN : in\_ord(p_3) = 0$,

3. $\exists p_4 \in IN : in\_ord(p_4) = |IN| - 1 \wedge \forall p_5 \in IN : in\_ord(p_5) \leq in\_ord(p_4)$,

potom relace in_ord jednoznačně identifikuje každou vstupní bránu pomocí čísla z množiny přirozených čísel.

První pravidlo definuje, že každé vstupní bráně je přiřazeno unikátní číslo z množiny přirozených čísel (včetně 0). Druhé pravidlo definuje, že nejmenší přiřazené číslo je 0. Dle druhého pravidla totiž musí být 0 přiřazena. Množina přirozených čísel již žádné menší číslo než 0 neobsahuje, a proto musí být všechna ostatní přiřazená čísla vyšší než 0 (stejná být nemohou kvůli prvnímu pravidlu). Třetí pravidlo definuje, že nejvyšší přiřazené číslo je $|IN| - 1$. Relace je definována pro všechny prvky z množiny IN, tudíž z druhého a třetího pravidla vyplývá, že mezi čísly 0 až $|IN| - 1$ neexistuje číslo, které by nebylo přiřazeno některé vstupní bráně.

Příklad 6.5.7:
$PI = \{A, B, Clk\}$
$IN = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}\}$

$in\_ord = \{(D_{R_1}, 0), (A_{ADD_1}, 1), (B_{ADD_1}, 2)\}$

V příkladu 6.5.7 je uvažován stejný obvod jako v příkladu 6.5.2. Vstupním branám budou dle definice relace *in_ord* přiřazena identifikační čísla 0, 1, 2. Dále je princip obdobný jako u příkladu 6.5.6. Jedno z možných přiřazení čísel vstupním branám lze vidět v příkladu 6.5.7.

6.5.10 Relace *out_ord*

Definice 6.5.10. *Nechť existuje relace* $out\_ord : OUT \rightarrow \aleph$, *která splňuje následující pravidla:*

1. $\forall p_1, p_2 \in OUT : p_1 \neq p_2 \rightarrow out\_ord(p_1) \neq out\_ord(p_2)$,

2. $\exists p_3 \in OUT : out\_ord(p_3) = 0$,

3. $\exists p_4 \in OUT : out\_ord(p_4) = |OUT| - 1 \wedge \forall p_5 \in OUT : out\_ord(p_5) \leq out\_ord(p_4)$,

potom relace out_ord jednoznačně identifikuje každou výstupní bránu pomocí čísla z množiny přirozených čísel.

První pravidlo definuje, že každé výstupní bráně je přiřazeno unikátní číslo z množiny přirozených čísel (včetně 0). Druhé pravidlo definuje, že minimální přiřazené číslo je vždy 0. Dle druhého pravidla totiž musí být 0 přiřazena. Množina přirozených čísel již žádné menší číslo než 0 neobsahuje, a proto musí být všechna ostatní přiřazená čísla vyšší než 0 (stejná být nemohou kvůli prvnímu pravidlu). Třetí pravidlo definuje, že nejvyšší přiřazené číslo je vždy $|OUT| - 1$. Relace je definována pro všechny prvky z množiny OUT, tudíž z druhého a třetího pravidla vyplývá, že mezi čísly 0 až $|OUT| - 1$ neexistuje číslo, které by nebylo přiřazeno některé výstupní bráně.

Příklad 6.5.8:
$OUT = \{Q_{R_1}, Q_{ADD_1}\}$

$out\_ord = \{(Q_{R_1}, 0), (Q_{ADD_1}, 1)\}$

V příkladu 6.5.8 je uvažován stejný obvod jako v příkladu 6.5.2. Výstupním branám budou dle definice relace *out_ord* přiřazena identifikační čísla 0, 1. Dále je princip obdobný jako u příkladu 6.5.6. Jedno z možných přiřazení čísel vstupním branám lze vidět v příkladu 6.5.8.

6.5.11 Relace *driver*

Definice 6.5.11. *Nechť existuje relace* $driver : IN \rightarrow OUT \cup PI$*, která splňuje následující pravidlo:*

1. $\forall p_1 \in IN : p_1 \rightarrow p_2 \wedge p_2 \in OUT \cup PI \wedge (p_2, p_1) \in C$,

potom relace driver přiřazuje každé vstupní bráně odpovídající budící bránu.

V modelu je uvažována propojovací strategie multiplexovaných datových cest. U této strategie lze pro každou vstupní bránu identifikovat jednu tzv. budící bránu, která distribuuje logický signál na všechny brány k ní přípojené. Budící brána může být buď primární vstupní brána nebo výstupní brána některého obvodového prvku. Z tohoto důvodu je zavedena relace *driver*, která každé vstupní bráně přiřazuje odpovídající budící bránu.

Příklad 6.5.9:
$PI = \{A, B, Clk\}$
$IN = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}\}$
$OUT = \{Q_{R_1}, Q_{ADD_1}\}$

$driver = \{(D_{R_1}, Q_{ADD_1}), (A_{ADD_1}, A), (B_{ADD_1}, B)\}$

V příkladu 6.5.9 je uvažován stejný obvod jako v příkladu 6.5.2. Tedy pro bránu D_{R_1} je budící branou Q_{ADD_1}, pro bránu A_{ADD_1} je budící branou A, pro bránu B_{ADD_1} je budící branou B.

6.5.12 Scan řetězec SR_i

Definice 6.5.12. *Nechť* $n_s \geq 0$ *je celé číslo určující počet scan řetězců v obvodu a nechť* $n_r \in \aleph$ *a* $n_r \leq |SC|$ *a* $\exists s_1, s_2, \ldots, s_{n_r} \in SC$ *a* $i \in \aleph$ *a* $i > 0 \wedge i \leq n_s$*, potom uspořádaná* n_r*-tice* $SR_i = (s_1, s_2, \ldots, s_{n_r})$ *představuje scan řetězec* i*.*

SR_i definuje uspořádání scan řetězce i. Maximální možná délka scan řetězce je vymezena počtem scan registrů v množině $|SC|$, ale scan řetězec může být i kratší. V případě, že $SR_i = ()$, scan řetězec není využit. Pořadí prvků v SR_i je důležité, protože definuje pořadí zapojení jednotlivých registrů do řetězce scan.

Příklad 6.5.10:
$SC = \{R_1, R_2, R_3, R_4, R_5\}$

$SR_1 = (R_1, R_2)$
$SR_2 = (R_5, R_4, R_3)$

V příkladu 6.5.10 je uvažován obvod s pěti scan registry a dvěma řetězci scan. První scan řetězec SR_1 je složen z registrů R_1, R_2, jež jsou zapojeny v uvedeném pořadí, tedy $SR_1 = (R_1, R_2)$. Druhý scan řetězec SR_2 je složen z registrů R_5, R_4, R_3, jež jsou zapojeny v uvedeném pořadí, tedy $SR_2 = (R_5, R_4, R_3)$.

6.5.13 Topologie scan řetězců SRS

Definice 6.5.13. *Nechť $n_s \geq 0$ je celé číslo určující počet scan řetězců v obvodu a nechť existují scan řetězce SR_1, SR_2, SR_3,... SR_{n_s}, pro které platí $SR_1 \times SR_2 \times SR_3 \times \ldots \times SR_{n_s} \in SC^{|SC|}$, potom uspořádaná n_s-tice $SRS = (SR_1, SR_2, SR_3, \ldots SR_{n_s})$ představuje všechny scan řetězce v obvodu.*

Každý scan registr může být součástí nejvíce jednoho scan řetězce, a každý scan registr musí být přiřazen některému scan řetězci, což je v předchozí definici zajištěno podmínkou $SR_1 \times SR_2 \times SR_3 \times \ldots \times SR_n \in SC^{|SC|}$. Princip je demonstrován v příkladu 6.5.11. V případě, že $n_s = 0$, tak $SRS = ()$ a obvod neobsahuje žádný scan řetězec.

Příklad 6.5.11:
$SC = R_1, R_2, R_3$

$SRS = ((R_1, R_2), (R_3))$

Obvod v příkladu 6.5.11 obsahuje dva scan řetězce. První scan řetězec sestává z registrů R_1, R_2 zapojených v uvedeném pořadí, druhý sestává z registru R_3, tedy $SRS = ((R_1, R_2), (R_3))$. Pokud budeme při ověření postupovat dle definice 6.5.13, tak $(R_1, R_2) \times (R_3)$ lze zapsat jako $((R_1, R_2), (R_3))$ nebo jako (R_1, R_2, R_3), což jsou sémanticky shodné zápisy. Dále $SC^3 = (R_1, R_2, R_3), (R_1, R_3, R_2), (R_3, R_1, R_2), (R_3, R_2, R_1), (R_2, R_1, R_3), (R_2, R_3, R_1)$ a je tedy zjevné, že $(R_1, R_2, R_3) \in SC^3$, čímž je podmínka z definice 6.5.13 splněna. V případě, že by některý scan registr nebyl přiřazen scan řetězci, tedy např. R_2 by nebyl přiřazen, potom bychom po aplikaci kartézského součinu na jednotlivé scan řetězce obdrželi (R_1, R_3), čímž by podmínka z definice SRS nebyla splněna, protože $(R_1, R_3) \notin SC^3$. V případě, že by některý scan registr byl přiřazen více scan řetězcům, např. R_2 by byl přiřazen oběma scan řetězcům, potom bychom po aplikaci kartézského součinu na jednotlivé scan řetězce dostali (R_1, R_2, R_2, R_3) nebo (R_1, R_2, R_3, R_2) (dle pořadí zapojení R_2 do druhého scan řetězce), čímž by podmínka z definice 6.5.13 taktéž nebyla splněna, protože $(R_1, R_2, R_2, R_3) \notin SC^3$ a $(R_1, R_2, R_3, R_2) \notin SC^3$.

6.5.14 Relace $inscan\_ord$

Definice 6.5.14. *Nechť existuje relace $inscan\_ord : SC \rightarrow \aleph$, která splňuje následující pravidlo:*

1. $\forall s \in SC : \exists p \in in(s) : inscan\_ord(s) = in\_ord(p)$,

potom relace inscan_ord přiřazuje datové vstupní bráně každého scan registru identifikační číslo z množiny přirozených čísel.

Předpokládá se, že existuje vždy jen jedna datová vstupní brána scan registru pro paralelní načítání odezvy obvodu. Relace *inscan_ord* je tvořena složenou relaci *in_ord* po *in*, kde z množiny získané jako výsledek *in* se použije jen jedna brána. Relace je zavedena kvůli úspornějšímu zápisu v dále definovaných algoritmech.

Příklad 6.5.12:
$E = \{R_1, ADD_1\}$
$SC = \{R_1\}$
$SRS = ((R_1))$
$IN = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}\}$
$in = \{(R_1, \{D_{R_1}\}), (ADD_1, \{A_{ADD_1}, B_{ADD_1}\})\}$
$in\_ord = \{(D_{R_1}, 0), (A_{ADD_1}, 1), (B_{ADD_1}, 2)\}$

$inscan\_ord = \{(R_1, 0)\}$

V příkladu 6.5.12 je uvažován stejný obvod jako v příkladu 6.5.2. Do tohoto obvodu byl vložen jeden scan řetězec tvořený registrem R_1. Obsah E, SC, SRS, IN, in, in_ord lze odvodit z předchozích příkladů. Brána D_{R_1} je vstupní branou R_1, protože $(R_1, \{D_{R_1}\}) \in in$. Vstupní brána D_{R_1} má přiděleno identifikační číslo 0, protože $(D_{R_1}, 0) \in in\_ord$. Registr R_1 má tedy vstupní bránu s přiděleným identifikačním číslem 0, protože $inscan\_ord = \{(R_1, 0)\}$.

6.5.15 Relace *outscan_ord*

Definice 6.5.15. *Nechť existuje relace* $outscan\_ord : SC \rightarrow \aleph$*, která splňuje následující pravidlo:*

1. $\forall s \in SC : \exists p \in out(s) : outscan\_ord(s) = out\_ord(p)$,

potom relace outscan_ord přiřazuje datové výstupní bráně každého scan registru identifikační číslo z množiny přirozených čísel.

Předpokládá se, že existuje jen jedna datová výstupní brána scan registru pro paralelní vkládání testovacích vektorů do obvodu. Relace *outscan_ord* je tvořena složenou relaci *out_ord* po *out*, kde z množiny získané jako výsledek *out* se použije jen jedna brána. Relace je zavedena kvůli úspornějšímu zápisu v dále definovaných algoritmech.

Příklad 6.5.13:
$E = \{R_1, ADD_1\}$
$SC = \{R_1\}$
$SRS = ((R_1))$
$OUT = \{Q_{R_1}, Q_{ADD_1}\}$
$out = \{(R_1, \{Q_{R_1}\}), (ADD_1, \{Q_{ADD_1}\})\}$
$out\_ord = \{(Q_{R_1}, 0), (Q_{ADD_1}, 1)\}$

$outscan\_ord = \{(R_1, 0)\}$

V příkladu 6.5.13 je uvažován stejný obvod jako v příkladu 6.5.2. Do tohoto obvodu byl vložen jeden řetězec scan tvořený registrem R_1. Obsah E, SC, SRS, OUT, out, out_ord lze odvodit z předchozích příkladů. Brána Q_{R_1} je výstupní branou R_1, protože $(R_1, \{Q_{R_1}\}) \in out$. Výstupní brána Q_{R_1} má přiděleno identifikační číslo 0, protože $(Q_{R_1}, 0) \in out\_ord$. Registr R_1 má tedy výstupní bránu s přiděleným identifikačním číslem 0, protože $outscan\_ord = \{(R_1, 0)\}$.

6.5.16 Scan vektor SV_i

Definice 6.5.16. *Nechť $n_{sv} \geq 0$ je celé číslo označující počet scan vektorů, ze kterých sestává test a nechť $n_s = |SRS|$ je celé číslo označující počet scan řetězců v obvodu a nechť $i \in \aleph$ a $i > 0 \wedge i \leq n_{sv}$ a platí následující pravidla:*

1. $\forall j \in \aleph : j > 0 \wedge j \leq n_{sv} \rightarrow \exists SR_j \in SRS$,
2. $\forall k_j \in \aleph : k_j > 0 \wedge k_j \leq |SR_j| \rightarrow \exists v_{j,k_j} \in D \backslash \{\updownarrow\}$,

potom $SV_i = ((v_{1,1}, v_{1,2}, \ldots, v_{1,|SR_1|}), (v_{2,1}, v_{2,2}, \ldots, v_{2,|SR_2|}), \ldots, (v_{n_s,1}, v_{n_s,2}, \ldots, v_{n_s,|SR_{n_s}|}))$ je scan vektor i.

SV_i představuje hodnoty, které budou sériově načteny do jednotlivých scan registrů všech scan řetězců během testového cyklu i. První pravidlo definuje, že v obvodu musí být možné identifikovat jednotlivé scan řetězce SR_j. Druhé pravidlo definuje, že do jednotlivých scan registrů mohou být načítány jen statické hodnoty z oboru hodnot D. V případě nulové délky některého scan řetězce nebo nulového počtu scan registrů v některém scan řetězci, není obsah SV_i definován.

Příklad 6.5.14:
$SRS = ((R_1, R_2), (R_3))$

$SV_1 = ((0, 1), (1))$
$SV_2 = ((1, 0), (1))$

Obvod v příkladu 6.5.14 obsahuje dva scan řetězce, první sestávající z registrů R_1, R_2 v uvedeném pořadí, druhý sestávající z registru R_3. SV_1 definuje scan vektor číslo 1, který bude sériově načten do scan registrů. Po dokončení fáze i_1 (viz kapitola 6.2) bude tedy v registru R_1 načtena hodnota 0, v registru R_2 hodnota 1 a v registru R_3 hodnota 1. SV_2 definuje scan vektor číslo 2. Po dokončení fáze i_2 bude tedy v registru R_1 načtena hodnota 1, v registru R_2 hodnota 0 a v registru R_3 hodnota 1.

6.5.17 Posloupnost scan vektorů SVS

Definice 6.5.17. *Nechť $n_{sv} \geq 0$ je celé číslo označující počet scan vektorů, ze kterých sestává test a SV_1, SV_2, …, $SV_{n_{sv}}$ jsou jednotlivé scan vektory, potom $SVS = (SV_1, SV_2, \ldots, SV_{n_{sv}})$ značí, že jako první bude aplikován scan vektor SV_1, jako druhý scan vektor SV_2 a jako poslední scan vektor SV_3.*

Množina SVS definuje posloupnost aplikace scan vektorů.

Příklad 6.5.15:
$SRS = ((R_1, R_2), (R_3))$
$SV_1 = ((0, 1), (1))$
$SV_2 = ((1, 0), (1))$

$SVS = (SV_1, SV_2)$

V příkladu 6.5.15 je použit stejný scan řetězec a stejné dva scan vektory jako v příkladu 6.5.14. Scan vektory budou aplikovány v pořadí SV_1, SV_2, tedy $SVS = (SV_1, SV_2) = (((0, 1), (1)), ((1, 0), (1)))$.

6.5.18 Testovací vektor TV_i

Definice 6.5.18. *Nechť $n_{tv} \geq 0$ je celé číslo označující počet testovacích vektorů a nechť $n_{pi} = |PI|$ je celé číslo označující počet primárních vstupů obvodu a dále $v_1, v_2, \ldots, v_{n_{pi}} \in D$ a $i \in \aleph$ a $i > 0 \wedge i \leq n_{tv}$ potom $TV_i = (v_1, v_2, \ldots, v_{n_{pi}})$ je testovací vektor i.*

Většina číslicových obvodů obsahuje alespoň jeden primární datový vstup (mimo synchronizační vstupy). Pro větší obecnost jsou v modelu uvažovány i obvody bez primárního vstupu, což mohou být např. senzory či generátory náhodných čísel, generátory synchronizačního signálu. U těchto obvodů se předpokládá aplikace testovacích vektorů jen pomocí řetězce scan. Pokud tyto obvody neobsahují řetězec scan, nelze na ně navrhovanou metodiku aplikovat.

Generátor testovacích vektorů přiřadí ke každému testovacímu vektoru odpovídající vektor, jež odpovídá korektní odezvě obvodu na aplikovaný testovací vektor. Pokud obvod nemá primární výstupy, je v této práci uvažována pouze odezva sejmutá pomocí scan řetězce (řetězců). Konkrétní přiřazení odezev k testovacím vektorům není pro implementaci metodiky podstatné. Pro simulaci aplikace testu je totiž uvažován korektně fungující obvod, takže aktuální odezvy obvodu na konkrétní pořadí aplikace testovacích vektorů jsou automaticky získány během simulace. Avšak při programování ATE musí být odpovídající odezvy obvodu uvažovány v korektním pořadí.

Navázání primárních vstupů na jednotlivé bity testovacího vektoru je zajištěno pomocí relace $pin\_ord(pi)$.

Příklad 6.5.16:
$PI = \{A, B, Clk\}$
$pin\_ord = \{(A, 0), (B, 1), (Clk, 2)\}$

$TV_1 = (0, 1, \updownarrow)$
$TV_2 = (1, 1, \updownarrow)$

V příkladu 6.5.16 je použit stejný obvod jako v příkladu 6.5.2. Jsou uvažovány dva testovací vektory TV_1, TV_2. V případě prvního testovacího vektoru $TV_1 = (0, 1, \updownarrow)$, což značí, že na primární vstup A bude přivedena hodnota logická 0, protože logická 0 je v TV_1 na indexu 0 a zároveň $(A, 0) \in pin\_ord$. Na primární vstup B bude přivedena hodnota logická 1, protože logická 1 je v TV_1 na indexu 1 a zároveň $(B, 1) \in pin\_ord$. Na synchronizační vstup bude přiveden pulz, který umožní aplikaci testovacího vektoru. Pulz je představován symbolem $\updownarrow$ v TV_1. Protože $\updownarrow$ je v TV_1 na indexu 2 a zároveň $(Clk, 2) \in pin\_ord$ bude pulz

přiveden na primární vstup Clk. Obdobně pro TV_2, tedy na A bude přivedena hodnota logická 1, na B bude přivedena hodnota logická 1 a na Clk bude přiveden synchronizační pulz.

6.5.19 Testové cykly TVS

Definice 6.5.19. *Nechť $n_{tv} > 0$ je celé číslo označující počet testovacích vektorů a TV_1, TV_2, TV_3, ..., $TV_{n_{tv}}$ jsou testovací vektory vygenerované generátorem testovacích vektorů v uvedeném pořadí a nechť $n_{sv} = |SVS|$ je počet scan vektorů, ze kterých sestává test. Jestliže $n_{tv} = n_{sv}$ nebo $n_{sv} = 0$ a jiný případ není dovolen, potom $TVS = (TV_1, TV_2, TV_3, \ldots, TV_{n_{tv}})$ představuje jednotlivé cykly testu.*

Počet prvků v TVS musí být roven počtu testovacích vektorů. Počet scan vektorů musí být roven počtu testovacích vektorů nebo nulový (v případě kombinačních obvodů). Pokud obvod neobsahuje primární vstupy, musí být doplněny prázdné testovací vektory, tedy potom $TV_1 = TV_2 = \ldots = TV_{n_{sv}} = ()$. Aby byl test platný, musí být definován alespoň jeden testový cyklus. Počet simulovaných testových cyklů bude u obvodu obsahujícího plný scan roven $|TVS|+1$, protože závěrečný testový cyklus sloužící pro vyprázdnění scan řetězců není do TVS zahrnut. Tento závěrečný testový cyklus je totiž zahrnut do simulačních algoritmů, jež budou definovány v kapitole 6.9.

Příklad 6.5.17:
$PI = \{A, B, Clk\}$
$SRS = ((R_1, R_2))$
$SVS = (((0,1)), ((1,0)))$
$pin\_ord = \{(A,0), (B,1), (Clk,2)\}$

$TVS = ((0,1,\updownarrow), (1,1,\updownarrow))$

V příkladu 6.5.17 je uvažován obvod s plným scanem, který obsahuje jeden řetězec scan tvořený registry R_1, R_2 zapojenými v uvedeném pořadí. Obvod obsahuje tři primární vstupy A, B, Clk. Během testu budou aplikovány dva scan vektory a dva testovací vektory. Nejdříve bude sériově načten scan vektor $(0,1)$. Po načtení tohoto vektoru bude v R_1 hodnota logická 0 a v R_2 hodnota logická 1. Následně bude na primární vstupy aplikován testovací vektor $(0,1,\updownarrow)$. Tedy na primární vstup A bude nastavena hodnota logická 0, na primární vstup B bude nastavena hodnota logická 1 a na primární vstup Clk bude přiveden synchronizační pulz. Potom bude z R_1, R_2 sériově vyčtena odezva obvodu na testovací vektory a zároveň bude do scan řetězce sériově načten vektor $(1,0)$. Po načtení tohoto vektoru bude v R_1 hodnota logická 1 a v R_2 hodnota logická 0. Následně bude aplikován testovací vektor $(1,1,\updownarrow)$. Tedy na primární vstup A bude nastavena hodnota logická 1, na primárním vstupu B zůstane zachována hodnota logická 1 a na primární vstup Clk bude přiveden synchronizační pulz. Na závěr bude ze scan řetězce sériově vyčtena odezva obvodu na poslední testovací vektory.

6.5.20 Relace ζ

Definice 6.5.20. *Nechť existuje relace $\zeta : SC \times D \times D \times MT \rightarrow \Re$ která splňuje následující pravidlo:*

1. $\forall s \in SC, \forall b_1 \in D, \forall b_2 \in D, \forall m \in MT : \zeta(s, b_1, b_2, m) \geq 0$,

potom relace ζ definuje příkon scan registrů v jednotlivých příkonových metrikách.

Relace ζ přiřazuje každému scan registru $s \in SC$ reálné číslo odpovídající příkonu vyjádřenému v příkonové metrice $m \in MT$ v závislosti na aktuální hodnotě $b_2 \in D$ uložené ve scan registru a nové sériově načítané hodnotě $b_1 \in D$.

Příklad 6.5.18:
Nechť existuje scan registr R_1, který je v daném časovém okamžiku nastaven na hodnotu logická 0. Do tohoto scan registru má být sériově načtena nová hodnota 1, což lze zapsat jako: $\zeta(R_1, 1, 0, NTC) = a$, kde a je hodnota odpovídající počtu překlopení (v metrice NTC). Konkrétní hodnota a závisí na použité technologické knihovně. Tato relace má uplatnění v případě, že jsou použity „izolovatelné“ registry scan (viz 4.2.2). Relace je využita v simulačním algoritmu definovaném v kapitole 6.9.

6.5.21 Relace ν

Definice 6.5.21. *Nechť existuje relace $\nu : E \times D^{|IN|} \times D^{|OUT|} \rightarrow D^{|OUT|}$, potom ν buď přechodovou funkci pro jednotlivé prvky obvodu.*

Přechodová funkce je nejčastěji definovaná pomocí pravdivostních tabulek. Při přechodu je uvažován i předchozí stav výstupů obvodového prvku, aby bylo možné modelovat i sekvenční obvodové prvky, jako jsou např. registry.

Příklad 6.5.19:
$E = \{R_1, ADD_1\}$
$IN = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}\}$
$in = \{(R_1, \{D_{R_1}\}), (ADD_1, \{A_{ADD_1}, B_{ADD_1}\})\}$
$in\_ord = \{(D_{R_1}, 0), (A_{ADD_1}, 1), (B_{ADD_1}, 2)\}$
$OUT = \{Q_{R_1}, Q_{ADD_1}\}$
$out = \{(R_1, \{Q_{R_1}\}), (ADD_1, \{Q_{ADD_1}\})\}$
$out\_ord = \{(Q_{R_1}, 0), (Q_{ADD_1}, 1)\}$

$\nu(ADD_1, (0, 0, 1), (0, 0)) = (0, 1)$

V příkladu 6.5.19 je uvažován stejný obvod jako v příkladu 6.5.2. Obsah E, IN, in, in_ord, OUT, out, out_ord lze tedy odvodit z předchozích příkladů. Tedy $\nu(ADD_1, (0, 0, 1), (0, 0))$ značí přechod prvku ADD_1 po nastavení vstupů na $(0, 0, 1)$ při stavu výstupů $(0, 0)$ do nového stavu, kdy bude na výstupech $(0, 1)$. Uspořádaná trojice $(0, 0, 1)$ představuje hodnoty nastavené na vstupy, tedy 0 na indexu 0 značí, že brána s přiřazeným číslem 0 bude nastavena na hodnotu logická 0. Na hodnotu logická 0 bude tedy nastavena brána D_{R_1}, protože $(Q_{R_1}, 0) \in in$ (odtud index 0). Nula na indexu 1 značí, že brána s přiřazeným číslem 1 bude nastavena na hodnotu logická 0. Na hodnotu logická 0 bude tedy nastavena i brána A_{ADD_1}, protože $(A_{ADD_1}, 1) \in in$ (odtud index 1). Jedna na indexu 2 značí, že brána s přiřazeným číslem 2 bude nastavena na hodnotu logická 1. Na hodnotu logická 1 bude tedy nastavena brána B_{ADD_1}, protože $(B_{ADD_1}, 2) \in in$ (odtud index 2). Vstupy A_{ADD_1}, B_{ADD_1} sčítačky ADD_1 jsou tak nastaveny na hodnoty 0, 1. Z pravdivostní tabulky pro sčítačku zjistíme, že výsledkem bude hodnota 1 na výstupu

sčítačky Q_{ADD_1}. Jiné prvky než ADD_1 nejsou při přechodu uvažovány. Vstupy všech ostatních obvodových prvků kromě ADD_1 jsou během přechodu zamaskovány, takže na hodnotě prvního prvku trojice $(0,0,1)$ nezáleží. Předchozí stav výstupů byl $(0,0)$, stav výstupů po přechodu tedy bude $(0,1)$. Nula na indexu 0 značí, že na výstupu Q_{R_1} zůstává hodnota 0, protože $(Q_{R_1},0) \in out$ (odtud index 0) a prvek R_1 nebyl během přechodu uvažován, takže zůstává nastavena původní hodnota. Jedna na indexu 1 značí, že na výstupu Q_{ADD_1} bude po přechodu hodnota 1, protože $(Q_{ADD_1},1) \in out$ (odtud index 1).

6.5.22 Relace η

Definice 6.5.22. *Nechť existuje relace* $\eta : E \times D^{|IN|} \times D^{|OUT|} \times MT \rightarrow \Re$, *která splňuje následující pravidlo:*

1. $\forall e \in E, \forall b_1 \in D^{IN}, \forall b_2 \in D^{OUT}, \forall m \in MT : \eta(e,b_1,b_2,m) \geq 0$,

potom relace η *kvantifikuje příkonovou metriku obvodových prvků při přechodu mezi jednotlivými stavy.*

Relace kvantifikuje příkonovou metriku $m \in MT$ během přechodu každého obvodového prvku mezi jednotlivými stavy. Tato relace bývá nejčastěji definovaná pomocí tabulky a závisí na použité technologické knihovně.

Příklad 6.5.20:
$E = \{R_1, ADD_1\}$
$IN = \{D_{R_1}, A_{ADD_1}, B_{ADD_1}\}$
$in = \{(R_1, \{D_{R_1}\}), (ADD_1, \{A_{ADD_1}, B_{ADD_1}\})\}$
$in\_ord = \{(D_{R_1},0), (A_{ADD_1},1), (B_{ADD_1},2)\}$
$OUT = \{Q_{R_1}, Q_{ADD_1}\}$
$out = \{(R_1, \{Q_{R_1}\}), (ADD_1, \{Q_{ADD_1}\})\}$
$out\_ord = \{(Q_{R_1},0), (Q_{ADD_1},1)\}$

$\eta(ADD_1, (0,0,1), (0,0), NTC) = a$

V příkladu 6.5.20 je uvažován stejný obvod jako v příkladu 6.5.2. Tedy např. $\eta(ADD_1, (0,0,1), (0,0), NTC)$ kvantifikuje příkonovou metriku NTC při přechodu prvku ADD_1 ze stavu definovaného stavem výstupů $(0,0)$ do stavu po nastavení vstupů na hodnoty $(0,0,1)$. Výsledkem je hodnota a. Konkrétní číselná hodnota závisí na použité realizační technologii. Princip je obdobný jako v příkladu 6.5.19. V případě registrů je přechodem míněna událost po příchodu synchronizačního pulzu.

6.6 Operátory a pomocné algoritmy

Nyní budou definovány operátory a pomocné algoritmy, které budou dále využity v kapitolách 6.9, 6.8 při definici simulačních algoritmů a algoritmů týkajících se GA. Jmenovitě bude definován operátor *max* pro výběr největšího prvku množiny, operátory *car*, *cdr*, *push* pro operace s posloupnostmi prvků, algoritmy *len*, *subset*, *get*, *replace*, *get*2*d*, *replace*2*d* pro základní práci s posloupnostmi, algoritmy *init_seq*, *init_scan* pro inicializaci posloupností a algoritmy *swap*, *sort* pro řazení posloupností. Tyto operátory a algoritmy pracují s definovaným formálním modelem. V algoritmech budou často využívány proměnné. Pro

přiřazení hodnoty do proměnné bude používán operátor přiřazení „:=“. V některých případech bude používán pojem *algoritmus funkce*. Tento pojem je použit pro označení algoritmu podprogramu, jehož výsledkem je hodnota implicitně předávaná volajícímu programu (výsledek funkce), takže volání podprogramu může být užito i v matematických výrazech jako funkce. Protože uvažujeme reálné obvody, jež obsahují konečné množství prvků, je pro další definice možné omezit u GA alelu B z obecné definice (viz definice 6.3.1) na konečnou podmnožinu přirozených čísel, aniž by došlo k ovlivnění kvality výsledků.

6.6.1 Operátor *max*

Operátor *max* slouží pro výběr největšího prvku z množiny.

Definice 6.6.1. *Nechť $B \subseteq \aleph$, B je konečná množina, nad B jsou definovány operátory porovnání a platí $\forall b \in B : b \leq max\ B$, potom $max\ B$ je největší prvek z B.*

6.6.2 Operátor *car*

Operátor *car* vrací první prvek posloupnosti.

Definice 6.6.2. *Nechť existuje posloupnost A, $A = (a_1, a_2, \ldots, a_n)$, kde $n \geq 1$, potom $car\ A = a_1$.*

Operátor *car* je definován pro posloupnosti obsahující jeden nebo více prvků. Pro prázdnou posloupnost není operátor definován a při použití v algoritmech musí být tento případ explicitně ošetřen. Při implementaci se tento stav nedefinovaného výsledku operátoru zpravidla řeší vyvoláním programové výjimky. Příklady 6.6.1, 6.6.2, 6.6.3 demonstrují tři různé příklady použití operátoru *car*.

Příklad 6.6.1:
$A = (2, 3, 1)$

$car\ A = 2$

V příkladu 6.6.1 je operátor *car* aplikován na posloupnost A sestávající ze tří prvků 2, 3, 1 v uvedeném pořadí. Operátor $car\ A$ vrací první prvek posloupnosti, tedy prvek 2.

Příklad 6.6.2:
$B = (1)$

$car\ B = 1$

V příkladu 6.6.2 je použita posloupnost B obsahující jediný prvek 1. Operátor $car\ B$ vrací tedy prvek 1.

Příklad 6.6.3:
$C = ()$

$car\ C =$ nedef.

Příklad 6.6.3 demonstruje případ aplikace operátoru *car* na prázdnou posloupnost C. V tomto případě $car\ C$ není definováno.

6.6.3 Operátor *cdr*

Operátor *cdr* vrací posloupnost bez prvního prvku původní posloupnosti.

Definice 6.6.3. *Nechť existuje posloupnost* A, $A = (a_1, a_2, \ldots, a_n)$, *kde* $n \geq 2$ *a nechť existuje posloupnost* B, $B = (a_1)$ *a nechť existuje prázdná posloupnost* C, $C = ()$, *potom* $cdr\ A = (a_2, \ldots, a_n)$ *a* $cdr\ B = ()$ *a* $cdr\ C = ()$.

Operátor *cdr* je definován i pro prázdnou posloupnost. Operátor *cdr* spolu s operátorem *car* tvoří základní operátory pro procházení posloupností. Příklad 6.6.4 demonstruje typickou kombinaci použití obou operátorů.

Příklad 6.6.4:
$A = (3, 2, 1)$

$car\ cdr\ A = 2$

V příkladu 6.6.4 je operátor *car* aplikován po operátoru *cdr*, který je aplikován na posloupnost A. Posloupnost A sestává ze tří prvků 3, 2, 1 v uvedeném pořadí. Operátor $cdr\ A$ vrací posloupnost $(2, 1)$. Na tuto posloupnost je dále aplikován operátor *car*. Dále $car\ (2, 1)$ vrací první prvek posloupnosti $(2, 1)$, což je 2 a to je také výsledek celého složeného výrazu.

6.6.4 Operátor *push*

Binární operátor *push* přidává prvek na konec posloupnosti.

Definice 6.6.4. *Nechť existuje posloupnost* A, $A = (a_1, a_2, \ldots, a_n)$, *kde* $n \geq 1$ *a nechť existuje prázdná posloupnost* B, $B = ()$, *potom* $A\ push\ a = (a_1, a_2, \ldots, a_n, a)$ *a* $B\ push\ a = (a)$.

Operátor *push* slouží pro přidání prvku na konec posloupnosti. Umožňuje vložit prvek i do prázdné posloupnosti. Operátor *push* je binární operátor. Není komutativní ani asociativní. V příkladu 6.6.5 lze vidět možnost použití operátoru *push* pro vložení dvou prvků do prázdné posloupnosti.

Příklad 6.6.5:
$A = ()$

$(A\ push\ 2)\ push\ 1 = (2, 1)$

V příkladu 6.6.5 je A prázdná posloupnost. Výsledkem $A\ push\ 2$ je jednoprvková posloupnost (2). Výsledkem (2) *push* 1 je tedy posloupnost (2, 1).

6.6.5 Algoritmus funkce $len(A)$

Funkce $len(A)$ vrací délku posloupnosti A.

Vstupem je posloupnost A. Návratovou hodnotou je kladná celočíselná hodnota odpovídající délce posloupnosti A.

Algoritmus $len(A)$

```
1 i := 0
2 AA := A
3 Dokud AA ≠ () prováděj:
4     AA := cdr AA
5     i := i + 1
6 Vrať i
```

Popis algoritmu $len(A)$**:** Na řádku 1 se inicializuje pomocná proměnná i pro počítání počtu prvků na 0. Na řádku 2 se kopíruje vstupní posloupnost A do pomocné proměnné AA, aby nedošlo k modifikaci A. Na řádcích 3 až 5 je zapsán cyklus, který počítá počet prvků v posloupnosti. Cyklus končí, jakmile dojde k vyprázdnění posloupnosti AA. V těle cyklu je vždy odebrán první prvek z posloupnosti AA pomocí operátoru cdr (řádek 4) a zvýšeno počítadlo prvků o 1 (řádek 5). Hodnota počítadla počtu prvků je vrácena volajícímu programu jako implicitní hodnota na řádku 6.

V příkladu 6.6.6 je demonstrována aplikace funkce len na tříprvkovou posloupnost.

Příklad 6.6.6:
$A = (2, 3, 1)$

$len(A) = 3$

V příkladu 6.6.6 je A posloupnost prvků 2, 3, 1 v uvedeném pořadí. Výsledkem $len(A)$ je hodnota 3 odpovídající počtu prvků v posloupnosti A.

6.6.6 Algoritmus funkce $subset(A, k_1, k_2)$

Funkce *subset* vrací část původní posloupnosti.

Vstupem je posloupnost A, konstanta k_1 a konstanta k_2. Návratovou hodnotou je posloupnost prvků z A začínající prvkem s indexem k_1 a končící prvek s indexem $k_2 - 1$ včetně. Prvky jsou počítány od indexu 0. Pokud jsou indexy k_1, k_2 mimo rozsah posloupnosti A, je vrácena přiměřeně zkrácená posloupnost.

Algoritmus $subset(A, k_1, k_2)$

```
1  i := 0
2  AA := A
3  B := ()
4  Dokud i ≠ k1 a len(AA) ≠ 0 prováděj:
5      AA := cdr AA
6      i := i + 1
7  Dokud i < k2 a len(AA) ≠ 0 prováděj:
8      B push (car AA)
9      AA := cdr AA
10     i := i + 1
11 Vrať B
```

Popis algoritmu $subset(A, k_1, k_2)$**:** Na řádcích 1 až 3 probíhá inicializace algoritmu. Nejdříve je nastaveno počítadlo prvků i na hodnotu 0 (řádek 1), dále je zkopírována vstupní posloupnost A do pomocné proměnné AA, aby nedošlo k její modifikaci (řádek 2). Na řádku

3 je inicializována pomocná posloupnost B na prázdnou hodnotu. Do této proměnné bude ukládána nová posloupnost. Cyklus na řádcích 4 až 6 slouží pro postupné odstraňování prvků z posloupnosti AA až po prvek, jež má v původní posloupnosti index k_1. Zároveň je v ukončovací podmínce cyklu ošetřen případ, kdy je k_1 mimo rozsah vstupní posloupnosti (druhá podmínka cyklu). Pro odstranění prvku z posloupnosti AA je použit operátor *cdr* (řádek 5). Po odstranění prvku je inkrementováno počítadlo prvků i (řádek 6). Po ukončení cyklu posloupnost AA začíná prvkem, jež měl v původní posloupnosti index k_1. Nyní zbývá překopírovat $k_2 - k_1$ prvků z AA do B. K tomu slouží cyklus na řádcích 7 až 10. Prvky jsou postupně kopírovány pomocí kombinace operátorů *push* a *car* (řádek 8). Po zkopírování prvku je tento odebrán z posloupnosti AA pomocí operátoru *cdr* (řádek 9). Dále je inkrementováno počítadlo prvků (řádek 10). V ukončovací podmínce cyklu je ošetřen i případ, kdy by bylo k_2 nastaveno mimo rozsah vstupní posloupnosti (druhá podmínka cyklu). Na řádku 11 je vrácena posloupnost B volajícímu programu jako implicitní hodnota. Pomocí řídicích podmínek cyklů je ošetřen i případ nulové vstupní posloupnosti.

V příkladech 6.6.7, 6.6.8, 6.6.9 lze vidět tři případy aplikace funkce *subset* na posloupnost A obsahující prvky 2, 3, 1, 5 v uvedeném pořadí.

Příklad 6.6.7:
$A = (2, 3, 1, 5)$

$subset(A, 1, 3) = (3, 1)$

V příkladu 6.6.7 je požadováno vytvoření posloupnosti sestávající z prvků s indexy 1 až $3 - 1$ původní posloupnosti A. Výsledná posloupnost bude tedy rovna $(3, 1)$.

Příklad 6.6.8:
$A = (2, 3, 1, 5)$

$subset(A, 3, 5) = (5)$

V příkladu 6.6.8 je požadováno vytvoření posloupnosti sestávající z prvků s indexy 3 až $5 - 1$ posloupnosti A. Koncový index je tedy $5 - 1 = 4$, což je mimo rozsah posloupnosti A, jejíž maximální použitelný index je 3 (4 prvky s indexem začínajícím od 0). Výsledná posloupnost bude tedy tvořena prvky od prvku s indexem 3 až do konce původní posloupnosti, takže výsledkem bude jednoprvková posloupnost (5).

Příklad 6.6.9:
$A = (2, 3, 1, 5)$

$subset(A, 2, 2) = ()$

V příkladu 6.6.9 je požadováno vytvoření posloupnosti sestávající z prvků s indexy 2 až $2 - 1$ posloupnosti A. Koncový index je tedy $2 - 1 = 1$, což je nižší hodnota než má počáteční index, takže výsledkem bude prázdná posloupnost.

6.6.6.1 Algoritmus funkce $get(A, k)$

Funkce *get* vrací prvek vyskytující se na zvolené pozici posloupnosti.

Vstupem je posloupnost A a konstanta k. Návratovou hodnotou je prvek posloupnosti A obsažený na pozici s indexem k. Funkce není platná pro index k mimo rozsah vstupní posloupnosti A.

Algoritmus $get(A, k)$

```
1 Vrať car subset(A, k, k + 1)
```

Popis algoritmu $get(A, k)$**:** Algoritmus sestává z jednoho řádku. Využívá se kombinace operátoru *car* a funkce *subset*. Funkce *subset* je využita k získání jednoprvkové posloupnosti, která obsahuje požadovaný prvek. Tato posloupnost je dále transformována na samostatný prvek pomocí operátoru *car*. Díky operátoru *car* je však třeba před použitím funkce zajistit, aby index k nebyl mimo rozsah vstupní posloupnosti A.

V příkladech 6.6.10, 6.6.11 lze vidět dva případy aplikace funkce *get* na posloupnost A obsahující prvky 3, 2, 5 v uvedeném pořadí.

Příklad 6.6.10:
$A = (3, 2, 5)$

$get(A, 1) = 2$

V příkladu 6.6.10 je požadováno extrahování prvku s indexem 1 z posloupnosti A. Výsledkem tedy bude prvek 2.

Příklad 6.6.11:
$A = (3, 2, 5)$

$get(A, 3) =$ nedef.

V příkladu 6.6.11 je požadováno extrahování prvku s indexem 3 z posloupnosti A. Index 3 je mimo rozsah posloupnosti A, takže výsledek operace není definován (díky použití operátoru *car* v definici funkce *get*). Těmto případům je nutné se při aplikaci funkce *get* vyhnout, např. kontrolou rozsahu vstupního indexu. Principiálně je možné kontrolu rozsahu vložit přímo do definice algoritmu *get*, ale v takovém případě je otázkou, jakým způsobem ošetřit případy neplatného indexu. Buďto by bylo nutné rezervovat jednu z možných návratových hodnot pro tzv. neplatný prvek nebo informovat volající algoritmus o chybě alternativním způsobem. V obou případech se jedná o implementační záležitosti, a proto nejsou v definici algoritmu řešeny.

6.6.6.2 Algoritmus $replace(A, k, p)$

Algoritmus *replace* umožňuje nahradit libovolný prvek posloupnosti.

Vstupem je posloupnost A, konstanta k a prvek p. Výstupem je modifikovaná posloupnost A, kde prvek s indexem k je nahrazen prvkem p.

Algoritmus $replace(A, k, p)$

```
1 i := 0
2 AA := A
3 A := ()
```

```
4  Dokud (i ≠ k) ∧ (len(AA) ≠ 0) prováděj:
5      A push (car AA)
6      AA := cdr AA
7      i := i + 1
8  Jestliže len(AA) ≠ 0:
9      A push p
10     AA := cdr AA
11 Dokud len(AA) ≠ 0 prováděj:
12     A push (car AA)
13     AA := cdr AA
```

Popis algoritmu $replace(A, k, p)$**:** Na řádcích 1 až 3 je algoritmus inicializován. Počítadlo prvků i je nastaveno na 0 (řádek 1). Do pomocné proměnné AA je uložena posloupnost A (řádek 2), aby byl zachován její obsah. Posloupnost A je inicializována na prázdnou hodnotu, aby do ní mohly být postupně vkládány prvky nové posloupnosti (řádek 3). V cyklu na řádcích 4 až 7 jsou do A kopírovány prvky z AA předcházející prvku s indexem k. V řídicí podmínce cyklu je ošetřen i případ, kdy by byl index k mimo rozsah posloupnosti. Prvky jsou kopírovány pomocí kombinace operátorů *car* a *push* (řádek 5). Každý zkopírovaný prvek je z posloupnosti AA odstraněn pomocí operátoru *cdr* (řádek 6). Dále je inkrementováno počítadlo prvků (řádek 7). V podmínce na řádcích 8 až 10 je do A vložen prvek s hodnotou p, pokud je k v rozsahu původní posloupnosti. Vlastní vložení prvku je provedeno pomocí operátoru *push* (řádek 9). Posunutí na následující prvek v posloupnosti AA, aby bylo možné nakopírovat zbytek posloupnosti do A, je zajištěno pomocí operátoru *cdr* (řádek 10). Cyklus na řádcích 11 až 13 slouží pro nakopírování zbytku posloupnosti AA do A. Princip je shodný, jako u předchozího cyklu, jen už není potřeba inkrementovat počítadlo prvků.

V příkladech 6.6.12, 6.6.13 lze vidět dva případy aplikace funkce *replace* na posloupnost A obsahující prvky 3, 2, 5 v uvedeném pořadí.

Příklad 6.6.12:
$A = (3, 2, 5)$

$replace(A, 1, 4)$
$A = (3, 4, 5)$

V příkladu 6.6.12 je požadováno nahrazení prvku s indexem 1 v posloupnosti A za prvek s hodnotou 4. Výsledkem tedy bude posloupnost $(3, 4, 5)$.

Příklad 6.6.13:
$A = (3, 2, 5)$

$replace(A, 3, 4)$
$A = (3, 2, 5)$

V příkladu 6.6.13 je požadováno nahrazení prvku s indexem 3 v posloupnosti A za prvek s hodnotou 4. Index 3 je mimo rozsah posloupnosti A (tři prvky s indexem od 0), výsledkem tedy bude nemodifikovaná posloupnost $(3, 2, 5)$.

6.6.7 Algoritmus funkce $get2d(A, k_1, k_2)$

Funkce $get2d$ vrací prvek vyskytující se na zvolené pozici posloupnosti, přičemž se uvažuje posloupnost, jež má minimálně dva rozměry. Tyto dva rozměry posloupnosti jsou modelovány jako tabulka, přičemž prvky vnější posloupnosti tvoří řádky této tabulky. Každý tento prvek je dále posloupnost, jejíž prvky tvoří sloupce daného řádku. Prvky jsou tedy indexovány pomocí dvojrozměrného indexu ve tvaru k_1, k_2, kde k_1 představuje řádek a k_2 sloupec vícerozměrné posloupnosti. Funkce $get2d$ je definována kvůli jednoduššímu zápisu dalších algoritmů.

Vstupem je posloupnost A a konstanty k_1, k_2. Návratovou hodnotou je prvek posloupnosti A obsažený na pozici s indexem k_1, k_2. Funkce není platná pro index k_1, k_2 mimo rozsah vstupní posloupnosti A. Funkce není platná pro posloupnosti, jež mají méně než 2 rozměry.

Algoritmus $get2d(A, k_1, k_2)$

```
1 Vrať get(get(A, k1), k2)
```

Popis algoritmu $get2d(A, k_1, k_2)$**:** Algoritmus sestává z jednoho řádku. Využívá se dvojité aplikace funkce get. První aplikace funkce get vybere řádek k_1 posloupnosti A. Druhá aplikace funkce get vybere na tomto řádku prvek s indexem k_2. Díky použití funkcí get je třeba zajistit, aby A byla minimálně dvojrozměrná posloupnost a aby indexy k_1, k_2 nebyly mimo rozsah vstupní posloupnosti A.

V příkladu 6.6.14 je demonstrována aplikace funkce $get2d$ na dvojrozměrnou posloupnost A, která obsahuje dva řádky. První řádek (řádek s indexem 0) obsahuje posloupnost $(3, 2, 5)$, druhý řádek (řádek s indexem 1) obsahuje posloupnost $(1, 2)$.

Příklad 6.6.14:
$A = ((3, 2, 5), (1, 2))$

$get2d(A, 1, 0) = 1$

V příkladu 6.6.14 je požadováno extrahování prvku s indexem 1, 0 (druhý řádek, první sloupec) z posloupnosti A. Výsledkem tedy bude prvek 1.

6.6.8 Algoritmus $replace2d(A, k_1, k_2, p)$

Algoritmus $replace2d$ umožňuje nahradit libovolný prvek posloupnosti, jež má minimálně dva rozměry. Pro uvažovanou posloupnost platí vše, co bylo řečeno v kapitole 6.6.7. Algoritmus $replace2d$ je definován kvůli jednoduššímu zápisu dalších algoritmů.

Vstupem je posloupnost A, konstanty k_1, k_2 a prvek p. Výstupem je modifikovaná posloupnost A, kde prvek s indexem k_1, k_2 je nahrazen prvkem p.

Algoritmus $replace2d(A, k_1, k_2, p)$

```
1 Jestliže (k1 ≥ 0) ∧ (k1 < len(A)):
2     AA := get(A, k1)
3     replace(AA, k2, p)
4     replace(A, k1, AA)
```

Popis algoritmu $replace2d(A, k_1, k_2, p)$**:** Algoritmus využívá dvojí aplikace funkce *replace*. Na řádku 1 je provedena kontrola rozsahu. Pokud je index k_1 mimo rozsah, nic se neprovede. Pokud je index k_1 v rozsahu, je na řádku 2 do pomocné proměnné AA uložen řádek s indexem k_1 posloupnosti A. Na řádku 3 je v AA nahrazen prvek s indexem k_2 prvkem p. Pokud je k_2 mimo rozsah AA, nic se neprovede (z definice funkce *replace*). Nakonec je na řádku 4 nahrazen řádek posloupnosti A s indexem k_1 za obsah AA, čímž je algoritmus ukončen.

V příkladu 6.6.15 je demonstrována aplikace funkce *replace*2*d* na dvojrozměrnou posloupnost A, která obsahuje dva řádky. První řádek (řádek s indexem 0) obsahuje posloupnost $(3, 2, 5)$, druhý řádek (řádek s indexem 1) obsahuje posloupnost $(1, 2)$.

Příklad 6.6.15:
$A = ((3, 2, 5), (1, 2))$

$replace2d(A, 0, 1, 6)$
$A = ((3, 6, 5), (1, 2))$

V příkladu 6.6.15 je požadována náhrada prvku s indexem 0, 1 (první řádek, druhý sloupec) posloupnosti A prvkem 6. Výsledkem tedy bude posloupnost $((3, 6, 5), (1, 2))$.

6.6.9 Algoritmus $init\_seq(A, n, p)$

Algoritmus *init_seq* slouží pro inicializaci posloupnosti.

Vstupem je posloupnost A, konstanta n a prvek p. Výstupem je inicializovaná posloupnost A. Algoritmus inicializuje posloupnost A vymazáním všech prvků a n násobným vložením prvku p. Pokud je p funkce, každý vkládaný prvek bude získán voláním této funkce.

Algoritmus $init\_seq(A, n, p)$

```
1 A := ()
2 Pro x := 1 až n prováděj:
3     A push p
```

Popis algoritmu $init\_seq(A, n, p)$**:** Na řádku 1 jsou z posloupnosti A vymazány všechny prvky. V cyklu na řádcích 2 až 3 je do posloupnosti A postupně vkládán prvek p (n-krát) pomocí operátoru *push* (řádek 3).

V příkladu 6.6.16 je demonstrována aplikace algoritmu *init_seq* na neprázdnou posloupnost A.

Příklad 6.6.16:
$A = (1, 2)$

$init\_seq(A, 3, 0)$
$A = (0, 0, 0)$

V příkladu 6.6.16 je vstupem *init_seq* neprázdná posloupnost A. Tato je aplikací *init_seq* vymazána a dále je do ní třikrát vložena hodnota 0. Výsledkem tedy bude posloupnost (0, 0, 0).

6.6.10 Algoritmus $init\_scan(S, R, p)$

Algoritmus *init_scan* inicializuje posloupnost na dva rozměry tak, aby její topologie odpovídala topologii inicializační posloupnosti. Algoritmus bude dále využíván pro inicializaci obsahu registrů scan.

Vstupem je posloupnost S, dvojrozměrná posloupnost R a prvek p. Výstupem je inicializovaná posloupnost S. Algoritmus inicializuje posloupnost S na hodnoty p. Pokud je p funkce, je každý vkládaný prvek získán voláním této funkce. Po inicializaci topologie posloupnosti S odpovídá topologii posloupnosti R.

Algoritmus $init\_scan(S, R, p)$

```
1 S := ()
2 A := ()
3 Pro x := 0 až len(R) − 1 prováděj:
4     init_seq(A, len(get(R, x)), p)
5     S push A
```

Popis algoritmu $init\_scan(S, R, p)$**:** Na řádcích 1 až 2 je provedena inicializace algoritmu. Během inicializace je do S vložena prázdná posloupnost (řádek 1) a dále je do pomocné proměnné A vložena prázdná posloupnost (řádek 2). V cyklu na řádcích 3 až 5 jsou postupně procházeny všechny řádky posloupnosti R. Je zjištěna délka každého procházeného řádku pomocí $len(get(R, x))$ (řádek 4) a pomocná posloupnost A je inicializována na stejný počet prvků hodnotou p pomocí *init_seq* (řádek 4). Takto inicializovaná pomocná posloupnost A je vložena na konec posloupnosti S (řádek 5) pomocí operátoru *push*.

V příkladu 6.6.17 je demonstrováno použití algoritmu pro inicializaci obsahu scan registrů ve scan řetězcích dle topologie scan řetězců SRS (viz definice 6.5.13). Jsou uvažovány dva scan řetězce, první tvořený registry R_1, R_2, druhý tvořený registry R_3, R_4. Obsah těchto scan registrů bude modelován pomocí prvků posloupnosti S. Pomocí algoritmu *init_scan* bude obsah scan registrů inicializován na hodnotu 0.

Příklad 6.6.17:
$S = (1, 0)$
$SRS = ((R_1, R_2), (R_3, R_4))$

$init\_scan(S, SRS, 0)$
$SRS = ((R_1, R_2), (R_3, R_4))$
$S = ((0, 0), (0, 0))$

V příkladu 6.6.17 je S přednastaveno na $(1, 0)$ pro demonstraci inicializační funkce algoritmu. Po aplikaci algoritmu *init_scan* bude posloupnost S vymazána a inicializována na hodnotu 0 dle topologie posloupnosti SRS. Posloupnost SRS nebude modifikována. Výsledkem tedy bude přiřazení dvojrozměrné posloupnosti $((0, 0), (0, 0))$ do S.

6.6.11 Algoritmus $swap(A, k_1, k_2)$

Algoritmus *swap* slouží pro záměnu dvou prvků v posloupnosti.

Vstupem je posloupnost A a konstanty k_1, k_2. Výstupem je modifikovaná posloupnost A. Algoritmus provádí záměnu dvou prvků, jež se nacházejí na indexech k_1, k_2 posloupnosti A.

Algoritmus $swap(A, k_1, k_2)$

```
1  b := get(A, k1)
2  replace(A, k1, get(A, k2))
3  replace(A, k2, b)
```

Popis algoritmu $swap(A, k_1, k_2)$**:** Algoritmus využívá algoritmů *get* a *replace*. Jedná se o klasický algoritmus pro záměnu dvou prvků posloupnosti. Pro provedení záměny se využívá pomocné proměnné b, do která je pomocí funkce *get* na řádku 1 vložen prvek, jež se nachází na indexu k_1 posloupnosti A. Dále je prvek nacházející se na indexu k_1 posloupnosti A nahrazen prvkem nacházejícím se na indexu k_2. Pro náhradu je využito kombinace algoritmů *replace* a *get* (řádek 2). Nakonec je na index k_2 vložen obsah pomocné proměnné b pomocí algoritmu *replace* (řádek 3). Protože je v algoritmu použita funkce *get*, je nutné zajistit, aby indexy k_1 a k_2 byly v rozsahu posloupnosti A.

V příkladech 6.6.18, 6.6.19 lze vidět dva případy aplikace funkce *swap* na posloupnost A obsahující prvky 3, 2, 5 v uvedeném pořadí.

Příklad 6.6.18:
$A = (3, 2, 5)$

$swap(A, 0, 2)$
$A = (5, 2, 3)$

V příkladu 6.6.18 je požadována záměna prvků s indexy 0 a 2 v posloupnosti A. Výsledkem je tedy posloupnost $(5, 2, 3)$.

Příklad 6.6.19:
$A = (3, 2, 5)$

$swap(A, 1, 4)$
$A =$ nedef.

V příkladu 6.6.19 je požadována záměna prvků s indexy 1 a 4 v posloupnosti A. Index 4 je však mimo rozsah posloupnosti A (tři prvky s indexem od 0), výsledek operace tudíž není definován.

6.6.12 Algoritmus funkce $sort(A)$

Funkce vrací posloupnost indexů odpovídajících seřazené vstupní posloupnosti.

Vstupem je posloupnost A obsahující prvky, nad kterými je definován operátor porovnání. Návratovou hodnotou je posloupnost prvků představující indexy do A navýšené o 1. Prvky této posloupnosti jsou tedy v rozsahu 1 až $len(A)$. Po odečtení 1 pak posloupnost těchto prvků definuje pořadí prvků posloupnosti A, jež odpovídá vzestupně seřazené posloupnosti A. Pro názornou demonstraci viz příklad 6.6.20.

Algoritmus $sort(A)$

```
1  AA := A
2  I := ()
3  Pro i := 1 až len(AA) prováděj:
```

```
4       I push i
5   Pro x := 0 až len(AA) − 1 prováděj:
6       Pro y := 0 až len(AA) − 2 prováděj:
7           Jestliže get(AA, y) > get(AA, y + 1):
8               swap(AA, y, y + 1)
9               swap(I, y, y + 1)
10  Vrať I
```

Popis algoritmu $sort(A)$**:** Algoritmus vychází ze standardního bublinkového algoritmu pro řazení. Na řádcích 1 až 4 je algoritmus inicializován. Na řádku 1 je zkopírován obsah vstupní posloupnosti A do AA, aby při běhu algoritmu nedošlo k modifikaci A. Dále je vložena prázdná posloupnost do proměnné I (řádek 2), jež představuje posloupnost indexů. Cyklus na řádcích 3 až 4 slouží pro inicializaci posloupnosti indexů I. V tomto cyklu jsou pomocí operátoru *push* (řádek 4) do I postupně vkládány hodnoty narůstající o 1. Cyklus na řádcích 5 až 9 provádí řazení prvků posloupnosti AA. Pokud je prvek napravo od právě zpracovávaného prvku menší než právě zpracovávaný prvek (řádek 7), je provedena záměna prvků posloupnosti AA (řádek 8) a zároveň jsou zaměněny i odpovídající indexy v I (řádek 9). Po seřazení vstupní posloupnosti je posloupnost indexů vrácena volajícímu programu jako implicitní hodnota (řádek 10). Metoda bublinkového řazení byla zvolena pro demonstraci problematiky. Při implementaci je vhodné pro řazení použít nějakou rychlejší metodu (např. modifikovaný quicksort), jejímž výstupem bude seřazená posloupnost indexů.

V příkladu 6.6.20 lze vidět využití funkce *sort* pro seřazení vstupní tříprvkové posloupnosti A.

Příklad 6.6.20:
$A = (5, 2, 3)$

$sort(A) = (2, 3, 1)$

V příkladu 6.6.20 je řazena posloupnost A sestávající z prvků 5, 2, 3 v uvedeném pořadí. Uvedené posloupnosti přidělíme posloupnost indexů (1, 2, 3). Posloupnost indexů tedy bude začínat od 1. Po vzestupném seřazení posloupnosti A bychom získali posloupnost $(2, 3, 5)$, jíž odpovídá posloupnost indexů $(2, 3, 1)$, která značí, že pro seřazení posloupnosti A je nutné druhý prvek přemístit na první pozici, třetí prvek na druhou pozici a první prvek na poslední pozici.

6.7 Kódování problému

Při použití GA hraje důležitou roli volba vhodného kódování problému. Nevhodná volba kódování problému je nejčastější příčinou selhání GA. V případě GA jedinci populace nesou genetickou informaci, tzv. genotyp, do kterého jsou zakódována jednotlivá kandidátní řešení, tzv. fenotyp. Nad genotypem jsou aplikovány jednotlivé genetické operátory. Při použití GA je cílem získat kandidátní řešení určitých kvalit, takže musí být definován způsob pro kvantitativní ohodnocení kvality jednotlivých řešení. Vyhodnocování kvality řešení probíhá postupně. Nejdříve je provedena konverze z genotypu (množiny B, viz definice 6.3.1) jedince na fenotyp (množinu S). K této konverzi se využívá funkce Δ. Kvalita fenotypu je ohodnocena reálným číslem pomocí funkce Φ. Relace Δ^{-1} se zpravidla využívá pro

Tabulka 6.7: Příklad dvojblokového chromozómu

b_{i_1}	b_{i_2}	b_{i_3}	...	$b_{i_{k-1}}$	b_{i_k}	$b_{i_{k+1}}$	$b_{i_{k+2}}$	...	b_{i_l}

Tabulka 6.8: Příklad dvojblokového chromozómu pro CUA se třemi testovacími vektory a třemi scan registry.

b_{i_1}	b_{i_2}	b_{i_3}	b_{i_4}	b_{i_5}	b_{i_6}

konstrukci prvotní populace jedinců z nějakého známého neoptimálního řešení problému. Obecně Δ^{-1} nemusí být funkcí. V takovém případě se při konstrukci prvotní populace volí jedna z možností mapování fenotypu na genotyp.

Navržený způsob kódování problému umožňuje do genotypu současně kódovat pořadí aplikace testovacích vektorů i pořadí zapojení registrů do řetězce scan. Zvolený koncept je rozšiřitelný, takže je možné do genotypu kódovat i pořadí aplikace testovacích vektorů pro několik CUA současně (bylo úspěšně použito např. ve spojení s testovatelnými bloky viz [78]) nebo pro CUA obsahující několik řetězců scan. Princip kódování spočívá v rozdělení chromozómu $CH = (b_{i_1}, b_{i_2}, \ldots, b_{i_l})$, kde $b_{i_1}, b_{i_2}, \ldots, b_{i_l} \in B$, $CH \in N$, l je délka chromozómu (viz definice 6.3.1) na jednotlivé bloky, kde každý blok kóduje pořadí aplikace testovacích vektorů na určité CUA nebo pořadí zapojení scan registrů do určitého scan řetězce. Počet bloků v chromozómu je tedy roven součtu počtu CUA a počtu řetězců scan. Každý blok sestává z jednoho nebo více genů. V tabulce 6.7 je prezentován příklad dvojblokového chromozómu, jež je nejběžnější při praktické aplikaci metodiky na obvody s plným scanem sestávajícím z jednoho řetězce scan. První blok, tvořený kódovými sekvencemi b_{i_1} až $b_{i_{k-1}}$, je využit pro kódování pořadí aplikace testovacích vektorů na CUA. Druhý blok, tvořený kódovými sekvencemi b_{i_k} až b_{i_l}, je využit pro kódování pořadí zapojení registrů do řetězce scan.

Obsah jednotlivých bloků je kódován samostatně. Využívá se systému priorit, ve kterém každý gen zastupuje prioritu předem zvoleného objektu (testovacího vektoru nebo scan registru dle aktuálního bloku). Kódové sekvence, jimiž jsou jednotlivé geny tvořeny, musí být mezi sebou porovnatelné. Dostačující podmínkou je, aby byl nad kódovými sekvencemi definován operátor porovnání $>$ (větší než) nebo $<$ (menší než). Vzestupným seřazením jednotlivých kódových sekvencí (v každém bloku samostatně) potom získáme pořadí jednotlivých objektů. Pro demonstraci principu budeme uvažovat příklad z tabulky 6.8.

V tabulce 6.8 je uveden příklad dvojblokového chromozómu pro CUA se třemi testovacími vektory, jimž odpovídají geny tvořené kódovými sekvencemi b_{i_1} až b_{i_3} a se třemi registry scan, jimž odpovídají geny tvořené kódovými sekvencemi b_{i_4} až b_{i_6}.

V tabulce 6.9 jsou uvedeny všechny možnosti seřazení kódových sekvencí b_{i_1} až b_{i_3} a k nim odpovídající pořadí aplikace testovacích vektorů pro chromozóm z tabulky 6.8. Hodnota b_{i_1} určuje pořadí aplikace testovacího vektoru v_1, hodnota b_{i_2} pořadí aplikace testovacího vektoru v_2, hodnota b_{i_3} pořadí aplikace testovacího vektoru v_3. Vektor odpovídající kódové sekvenci s nižší hodnotou bude aplikován před vektorem odpovídajícím kódové sekvenci s vyšší hodnotou. Jako první bude aplikován vektor odpovídající kódové sekvenci s nejnižší hodnotou. Vektor odpovídající kódové sekvenci s nejvyšší hodnotou bude aplikován jako poslední. Tedy např. ve druhém řádku tabulky 6.9 lze vidět, že pokud je možné kódové sekvence b_{i_1} až b_{i_3} seřadit do neklesající posloupnosti $(b_{i_2}, b_{i_3}, b_{i_1})$, tedy $b_{i_2} \leq b_{i_3} \leq b_{i_1}$,

Tabulka 6.9: Všechny možnosti pořadí aplikace testovacích vektorů pro chromozóm z tabulky 6.8.

Seřazené kódové sekvence	Pořadí aplikace testovacích vektorů
$b_{i_1} \leq b_{i_2} \leq b_{i_3}$	v_3 po v_2 po v_1
$b_{i_2} \leq b_{i_3} \leq b_{i_1}$	v_1 po v_3 po v_2
$b_{i_1} \leq b_{i_3} \leq b_{i_2}$	v_2 po v_3 po v_1
$b_{i_3} \leq b_{i_1} \leq b_{i_2}$	v_2 po v_1 po v_3
$b_{i_3} \leq b_{i_2} \leq b_{i_1}$	v_1 po v_2 po v_3
$b_{i_2} \leq b_{i_1} \leq b_{i_3}$	v_3 po v_1 po v_2

Tabulka 6.10: Všechny možnosti pořadí zapojení registrů do řetězce scan pro chromozóm z tabulky 6.8.

Seřazené kódové sekvence	Pořadí zapojení scan registrů
$b_{i_4} \leq b_{i_5} \leq b_{i_6}$	sc_3 po sc_2 po sc_1
$b_{i_5} \leq b_{i_6} \leq b_{i_4}$	sc_1 po sc_3 po sc_2
$b_{i_4} \leq b_{i_6} \leq b_{i_5}$	sc_2 po sc_3 po sc_1
$b_{i_6} \leq b_{i_4} \leq b_{i_5}$	sc_2 po sc_1 po sc_3
$b_{i_6} \leq b_{i_5} \leq b_{i_4}$	sc_1 po sc_2 po sc_3
$b_{i_5} \leq b_{i_4} \leq b_{i_6}$	sc_3 po sc_1 po sc_2

bude jako první aplikován testovací vektor v_2, jako druhý v_3 a jako poslední v_1.

V tabulce 6.10 jsou uvedeny všechny možnosti seřazení kódových sekvencí b_{i_4} až b_{i_6} a k nim odpovídající pořadí zapojení registrů do řetězce scan pro chromozóm z tabulky 6.8. Hodnota b_{i_4} určuje pořadí zapojení scan registru sc_1, hodnota b_{i_5} pořadí zapojení scan registru sc_2, hodnota b_{i_6} pořadí zapojení scan registru sc_3. Scan registr odpovídající kódové sekvenci s nižší hodnotou bude zapojen před scan registrem odpovídajícím kódové sekvenci s vyšší hodnotou. Jako první bude zapojen scan registr odpovídající kódové sekvenci s nejnižší hodnotou. Scan registr odpovídající kódové sekvenci s nejvyšší hodnotou bude zapojen jako poslední. Tedy např. ve třetím řádku tabulky 6.10 lze vidět, že pokud je možné kódové sekvence b_{i_4} až b_{i_6} seřadit do neklesající posloupnosti $(b_{i_4}, b_{i_6}, b_{i_5})$, tedy $b_{i_4} \leq b_{i_6} \leq b_{i_5}$, bude jako první v řetězci scan zapojen scan registr sc_1, jako druhý sc_3, jako poslední sc_2.

V případě nejednoznačnosti (shody s více řádky v jednotlivých tabulkách), je nutno nějakým způsobem vybrat z několika různých možností řešení. Jak bylo zjištěno experimenty na reálných obvodech, způsob volby řádku tabulky v těchto nejednoznačných případech nemá vliv na kvalitu dosažených výsledků. Jmenovitě bylo ověřováno použití první, poslední a stochasticky zvolené shody.

6.8 Algoritmy pro převod genotypu na fenotyp a zpět

Využití tabulek uvedených v kapitole 6.7 je pro komplexní obvody nepraktické. Z tohoto důvodu budou definovány algoritmy pro přímý převod genotypu na fenotyp a zpět. Jmenovitě bude definován algoritmus *delta* realizující funkci Δ pro převod chromozómu na

pořadí jednotlivých objektů. Pro konstrukci prvotní populace je vhodné použít existující neoptimální řešení (vygenerované návrhovým systémem). Jen tak bude zaručeno, že řešení nalezená GA nebudou nikdy horší než známé neoptimální řešení. Z tohoto důvodu bude definován algoritmus *delta_inv* pro konstrukci chromozómu z pořadí objektů. Tento algoritmus volí jednu z možností relace Δ^{-1}. Vzhledem ke způsobu konstrukce prvotní populace (viz kapitola 6.11) způsob volby v tomto algoritmu výrazněji neovlivní dosažené výsledky.

Pro definici obou algoritmů bude využit formální model definovaný v kapitole 6.5 a operátory a pomocné algoritmy definované v kapitole 6.6. Oba algoritmy budou definovány pro chromozóm složený z libovolného počtu bloků za předpokladu, že první blok kóduje pořadí aplikace testovacích vektorů a ostatní bloky kódují pořadí zapojení registrů do scan řetězců. V algoritmech bude využívána posloupnost K, která bude sloužit pro definici způsobu rozdělení chromozómu na jednotlivé bloky. Posloupnost K je inicializována aplikací algoritmu *delta_inv* a dále je využívána algoritmem *delta*. Pro chromozóm uvedený v tabulce 6.8 bude $K = (3)$, což znamená, že druhý blok chromozómu začíná na indexu 3 (číslováno od 0). Pokud by bylo např. $K = (3, 4)$, znamenalo by to, že druhý blok (kódující první scan řetězec) bude začínat na indexu 3 a třetí blok (kódující druhý scan řetězec) bude začínat na indexu 4. Tudíž první scan řetězec by obsahoval jeden scan registr a druhý scan řetězec by obsahoval dva scan registry (protože délka chromozómu $l = 6$). Pro $K = ()$ se jedná o kombinační obvod.

6.8.1 Algoritmus $delta(CH, K, TA, SCS)$

Algoritmus generuje posloupnost definující pořadí aplikace testovacích vektorů a posloupnost definující pořadí zapojení registrů do řetězců scan z chromozómu. Algoritmus realizuje výpočet funkce Δ.

Vstupem je chromozóm CH a posloupnost K popisující rozložení bloků v chromozómu. Výstupem je posloupnost definující pořadí aplikace testovacích vektorů TA a dvojrozměrná posloupnost SCS definující pořadí zapojení registrů do řetězců scan.

Algoritmus $delta(CH, K, TA, SCS)$

```
1  TA := ()
2  SCS := ()
3  KK := K
4  k1 := 0
5  k2 := l
6  Jestliže len(KK) ≠ 0:
7      k2 := car KK

9  KK := cdr KK
10 TA := sort(subset(CH, k1, k2))

12 Dokud len(KK) ≠ 0 prováděj:
13     k1 := k2
14     k2 := car KK
15     KK := cdr KK
16     SCS push sort(subset(CH, k1, k2))

18 Jestliže len(K) ≠ 0:
19     k1 := k2
```

20 $k_2 := l$
21 $SCS\ push\ sort(subset(CH,\ k_1,\ k_2)$

Popis algoritmu $delta(CH, K, TA, SCS)$**:** Na řádcích 1 až 7 je algoritmus inicializován. Na řádku 1 je inicializována posloupnost TA určující pořadí aplikace testovacích vektorů. Na řádku 2 je inicializována dvojrozměrná posloupnost SCS určující pořadí zapojení scan registrů do scan řetězců. Obsah posloupnosti K je zkopírován do pomocné proměnné KK, aby nedošlo k její modifikaci (řádek 3). Pomocná proměnná k_1, která určuje index začátku aktuálně zpracovávaného bloku chromozómu, je nastavena na 0 (řádek 4). Pomocná proměnná k_2 určující index začátku následujícího bloku je na řádku 5 nastavena na hodnotu l (l je délka chromozómu, viz definice 6.3.1). Pokud je KK neprázdná posloupnost (tedy CUA obsahuje alespoň jeden řetězec scan) , je splněna podmínka na řádku 6 a k_2 je nastaveno na hodnotu prvního prvku z KK pomocí operátoru car (řádek 7). Z KK je odstraněn první prvek, aby nebyl později opětovně použit (řádek 9).

Nyní může být zpracován první blok, což je zajištěno na řádku 10 pomocí aplikace algoritmů *subset* a *sort*, čímž získáme posloupnost indexů TA definující pořadí aplikace jednotlivých testovacích vektorů. Cyklus na řádcích 12 až 16 slouží pro zpracování ostatních bloků kromě posledního. Nejdříve jsou posunuty ukazatele k_1 a k_2, na řádcích 13 a 14. Potom je odstraněn odpovídající prvek z KK (řádek 15). Na závěr cyklu je vytvořena posloupnost zapojení scan registrů daného scan řetězce pomocí aplikace algoritmů *subset* a *sort*. Vytvořená posloupnost je pomocí operátoru *push* přidána na konec posloupnosti SCS (řádek 16). Cyklus končí po vyprázdnění posloupnosti KK.

V případě, že má chromozóm více než jeden blok, tak ještě zbývá zpracovat poslední blok chromozómu. Počet bloků se ověřuje na řádku 18. Pokud má chromozóm více než jeden blok, jsou posunuty ukazatele k_1 a k_2 na poslední blok (řádky 19 a 20). Další blok již není, proto se k_2 nastavuje na hodnotu l (délka chromozómu). Dále je vytvořena posloupnost definující pořadí zapojení scan registrů posledního scan řetězce pomocí aplikace algoritmů *subset* a *sort*. Vytvořená posloupnost je následně přidána pomocí operátoru *push* na konec posloupnosti SCS (řádek 21).

V příkladu 6.8.1 je demonstrováno dekódování chromozómu CH pomocí algoritmu *delta*.

Příklad 6.8.1:
$CH = (12, 2, 8, 10, 20, 11, 5, 9)$
$K = (3, 6)$

$delta(CH, K, TA, SCS)$
$TA = (2, 3, 1)$
$SCS = ((1, 3, 2), (1, 2))$

V příkladu 6.8.1 je vstupem chromozóm CH, $CH = (12, 2, 8, 10, 20, 11, 5, 9)$ a posloupnost K určující rozdělení chromozómu CH na bloky, $K = (3, 6)$. Z obsahu K lze určit, že chromozóm CH sestává ze tří bloků, z nichž první začíná na pozici 0 (první prvek CH) a končí na pozici 2 (hodnota prvního prvku z $K - 1$). Tento blok kóduje pořadí aplikace tří $(2-0+1)$ testovacích vektorů v_1, v_2, v_3 (princip kódování viz tabulka 6.9). Dále chromozóm obsahuje dva bloky definující pořadí zapojení scan registrů ve dvou scan řetězcích. Celkově druhý blok začíná na pozici 3 (první prvek z K) a končí na pozici 5 (druhý prvek z $K - 1$), obsahuje tedy zakódované pořadí zapojení tří $(5 - 3 + 1)$ registrů scan $sc_{1,1}$, $sc_{1,2}$, $sc_{1,3}$ prvního scan řetězce (princip kódování viz tabulka 6.10). Poslední blok začíná na pozici 6

(druhý prvek z K) a končí na pozici 7 ($len(CH) - 1$). Obsahuje tedy zakódované pořadí dvou ($7-6+1$) registrů scan $sc_{2,1}$, $sc_{2,2}$ druhého scan řetězce (princip kódování viz tabulka 6.10).

Po aplikaci algoritmu *delta* dostáváme $TA = (2, 3, 1)$, což značí, že nejdříve bude aplikován testovací vektor v_2, potom v_3 a jako poslední v_1. Tento případ rozebereme blíže. První blok v CH je roven $(12, 2, 8)$. Nejnižší hodnotou je 2, která se nachází na indexu 1. Index 1 odpovídá testovacímu vektoru v_2 $(1 + 1 = 2)$, proto bude jako první aplikován testovací vektor v_2. Druhou nejnižší hodnotou je 8, která se nachází na indexu 2. Index 2 odpovídá testovacímu vektoru v_3 $(2 + 1 = 3)$, proto bude jako druhý aplikován testovací vektor v_3. Nejvyšší hodnota je 12, která se nachází na indexu 0. Index 0 odpovídá testovacímu vektoru v_1 $(0 + 1 = 1)$, proto bude jako poslední aplikován testovací vektor v_1.

Dále dostáváme $SCS = ((1, 3, 2), (1, 2))$, což značí, že v prvním řetězci scan bude jako první zapojen registr $sc_{1,1}$, jako druhý registr $sc_{1,3}$ a jako poslední registr $sc_{1,2}$. Ve druhém scan řetězci bude jako první zapojen registr $sc_{2,1}$ a jako poslední registr $sc_{2,1}$. Princip dekódování je shodný jako u testovacích vektorů.

6.8.2 Algoritmus $delta\_inv(TA, SCS, CH, K)$

Algoritmus generuje chromozóm z posloupnosti definující pořadí aplikace testovacích vektorů a z dvojrozměrné posloupností definující pořadí zapojení registrů do řetězců scan. Algoritmus realizuje výpočet relace jež je podmnožinou relace Δ^{-1}.

Vstupem je posloupnost TA definující pořadí aplikace testovacích vektorů a dvojrozměrná posloupnost SCS definující pořadí zapojení registrů do scan řetězců scan. Výstupem je chromozóm CH a posloupnost K popisující rozložení bloků v chromozómu.

Algoritmus $delta\_inv(TA, SCS, CH, K)$

```
1  i := 0
2  A := TA
3  S1 := SCS
4  CH := ()
5  K := ()
6  Dokud len(A) ≠ 0 prováděj:
7      CH push i
8      i := i + 1
9      A := cdr A
10 Dokud len(S1) ≠ 0 prováděj:
11     S2 = car S1
12     K push i
13     Dokud len(S2) ≠ 0 prováděj:
14         CH push i
15         i := i + 1
16         S2 := cdr S2
17     S1 := cdr(S1)
```

Popis algoritmu $delta\_inv(TA, SCS, CH, K)$**:** Na řádcích 1 až 5 je algoritmus inicializován. Nejdříve je nastaveno počítadlo prvků i na hodnotu 0 (řádek 1). Dále je zkopírována posloupnost TA do A (řádek 2) a dvojrozměrná posloupnost SCS do S_1 (řádek 3). Taktéž je inicializován chromozóm CH (řádek 4) a posloupnost K popisující rozložení bloků ve chromozómu (řádek 5). V cyklu na řádcích 6 až 9 je nastaven první blok chromozómu definující

pořadí aplikace testovacích vektorů na CUA. Do CH jsou postupně vkládány hodnoty i vzrůstající o 1 (řádky 7 a 8). Zároveň jsou odebírány prvky z A (řádek 9). Cyklus končí po vyčerpání všech prvků z A. Cyklus na řádcích 10 až 17 slouží pro nastavení bloků odpovídajících jednotlivým scan řetězcům. Do S_2 je zkopírována posloupnost definující pořadí zapojení scan registrů u právě zpracovávaného scan řetězce (řádek 11). Tento scan řetězec je vyjmut z posloupnosti scan řetězců na řádku 17, aby nebyl opakovaně zpracováván. Index začátku tohoto bloku v chromozómu CH je poznamenán do posloupnosti K (řádek 12). Dále na řádcích 13 až 16 následuje cyklus pro zpracování posloupnosti definující pořadí zapojení scan registrů u daného scan řetězce. Do CH jsou postupně vkládány hodnoty i vzrůstající o 1 (řádky 14 a 15). Zároveň jsou odebírány prvky z S_2 (řádek 16). Cyklus končí po vyčerpání všech prvků z S_2.

V příkladu 6.8.2 je demonstrováno vytvoření chromozómu pomocí algoritmu *delta_inv*.

Příklad 6.8.2:
$TA = (1, 2, 3)$
$SCS = ((1, 2, 3), (1, 2))$

$delta\_inv(TA, SCS, CH, K)$
$CH = (0, 1, 2, 3, 4, 5, 6, 7)$
$K = (3, 6)$

V příkladu 6.8.2 je vytvořen chromozóm CH pro tři testovací vektory v_1, v_2, v_3 vygenerované návrhovým systémem v uvedeném pořadí ($TA = (1, 2, 3)$) a pro dva scan řetězce. První scan řetězec sestává ze tří scan registrů $sc_{1,1}$, $sc_{1,2}$, $sc_{1,3}$ zapojených v uvedeném pořadí a druhý scan řetězec sestává ze dvou scan registrů $sc_{2,1}$, $sc_{2,2}$ zapojených v uvedeném pořadí, tedy $SCS = ((1, 2, 3), (1, 2))$. Po aplikaci algoritmu *delta_inv* dostáváme chromozóm $CH = (0, 1, 2, 3, 4, 5, 6, 7)$, kde první tři prvky tvoří blok popisující pořadí aplikace testovacích vektorů. Tudíž blok popisující první scan řetězec začíná na indexu 3 (počítáno od 0) , odtud první prvek K je 3. Druhý prvek K je 6, tedy blok popisující druhý scan řetězec začíná v CH od indexu 6. Mezi jednotlivými hodnotami v CH lze vidět jistou závislost (konstantní přírůstek o 1). Z tohoto důvodů byl při praktické implementaci algoritmu pro zvýšení diverzity jednotlivých kódových sekvencí chromozómu použit parametr pro volbu velikosti přírůstku.

6.9 Algoritmy pro simulaci aplikace testu

Pro stanovení příkonových metrik byla zvolena metoda simulace. Během simulace je uvažován signál ideálního obdélníkového průběhu a využívá se model nulového zpoždění signálu. Předpokládá se použití nekmitajících obvodů. Byly vytvořeny dva algoritmy pro simulaci aplikace testu. Algoritmus pwr_1 je výpočetně méně náročný, protože jsou uvažovány izolovatelné řetězce scan (viz kapitola 4.2.2) a nepředpokládá se použití vycpávky. Při splnění těchto předpokladů není třeba simulovat sériové posuny vektorů scan řetězci a pro výpočet lze použít rovnic definovaných v kapitole 3.1.5. Tímto způsobem lze dosáhnout urychlení simulace. Algoritmus pwr_2 používá kompletní simulaci. Během simulace je uvažována vycpávka scan řetězců, takže jsou pokryty i případy, kdy obvod obsahuje scan řetězce různých délek. Simulace probíhá v krocích po jednotlivých synchronizačních pulzech. Přechodové stavy nejsou uvažovány.

Algoritmy definované v této kapitole využívají dříve definovaný formální model, operátory, pomocné algoritmy a funkce. Jmenovitě budou v této kapitole definovány algoritmy pwr_{scan}, $update$, pwr_1, $scan\_shift$, pwr_2. Algoritmus pwr_{scan} aplikuje rovnice definované v kapitole 3.1.5 pro stanovení příkonových metrik izolovatelných scan řetězců. Algoritmus $update$ simuluje přenos hodnot z výstupů obvodových prvků a primárních vstupů na vstupy k nim připojených obvodových prvků. Algoritmus je aplikován mezi jednotlivými kroky simulace kvůli aktualizaci hodnot na vstupních branách jednotlivých obvodových prvků. Algoritmus pwr_1 provádí zjednodušenou simulaci aplikace testu. Algoritmus $scan\_shift$ simuluje posun scan vektorů jednotlivými scan řetězci. Algoritmus pwr_2 provádí úplnou simulaci aplikace testu.

6.9.1 Algoritmus funkce $pwr_{scan}(SV_i, SVO_i, M)$

Funkce aplikuje rovnice definované v kapitole 3.1.5 pro stanovení příkonových metrik izolovatelných scan řetězců během sériového načítání scan vektorů a současného vyčítání předchozí odezvy obvodu.

Vstupem je dvojrozměrná posloupnost SV_i představující jeden sériově načítaný scan vektor pro jednotlivé scan řetězce ($SV_i \in SVS$), dvojrozměrná posloupnost SVO_i představující odpovídající sériově vyčítanou odezvu obvodu, jež je získána simulací (viz definice algoritmu 6.9.3), příkonová metrika $M \in \{NTC, WNTC, WSA\}$. Návratovou hodnotou je reálné číslo kvantifikující požadovanou příkonovou metriku.

Algoritmus $pwr_{scan}(SV_i, SVO_i, M)$

```
1  P := 0.0
2  Pro x := 0 až len(SV_i) − 1 prováděj:
3      SI := get(SV_i, x)
4      SO := get(SVO_i, x)
5      Jestliže M = NTC:
6          P := NTC_sc(SI, SO)
7      jinak: Jestliže M = WNTC:
8          P := WNTC_sc(SI, SO)
9      jinak: Jestliže M = WSA:
10         P := WSA_sc(SI, SO)
11 Vrať P
```

Popis algoritmu $pwr_{scan}(SV_i, SVO_i, M)$**:** Na řádku 1 je inicializována pomocná proměnná P na hodnotu 0.0. Na řádcích 2 až 10 se nachází hlavní část algoritmu sestávající z cyklu. Cyklus probíhá pro všechny scan řetězce. Na řádcích 3 a 4 jsou do SI a SO uloženy posloupnosti pro právě zpracovávaný scan řetězec. Dále je do proměnné P přiřazen výsledek výpočtu odpovídající rovnice z kapitoly 3.1.5. Předpokládá se $MT = \{NTC, WNTC, WSA\}$, jiné metriky nejsou podporovány. V případě požadavku na jinou metriku, zůstává proměnná P nastavena na hodnotu 0.0. Hodnota P je vrácena volajícímu programu jako implicitní hodnota na řádku 11.

6.9.2 Algoritmus $update(VTV, VO, VI)$

Algoritmus simuluje přenos hodnot z výstupů obvodových prvků a primárních vstupů obvodu na vstupy k nim připojených obvodových prvků.

Vstupem algoritmu je posloupnost VTV, která představuje hodnoty nastavené na primárních vstupech obvodu, posloupnost VO, která představuje hodnoty nastavené na jednotlivých výstupních branách obvodových prvků. Výstupem je posloupnost VI, která představuje hodnoty detekovatelné na vstupních branách jednotlivých obvodových prvků v ustáleném stavu (hodnoty nastavené odpovídajícími budiči). Předpokládá se, že všechny posloupnosti mají korektní velikost, tedy $len(VTV) = |PI|$, $len(VI) = |IN|$, $len(VO) = |OUT|$. Posloupnosti VTV, VO, VI jsou inicializovány a udržovány během simulace, viz definice algoritmů 6.9.3, 6.9.5.

Algoritmus $update(VTV, VO, VI)$

```
1 Pro ∀ in : in ∈ IN prováděj:
2     d := driver(in)
3     Jestliže d ∈ OUT:
4         replace(VI, in_ord(in), get(VO, out_ord(d)))
5     jinak:
6         replace(VI, in_ord(in), get(VTV, pin_ord(d)))
```

Popis algoritmu $update(VTV, VO, VI)$**:** Hlavní smyčka algoritmu probíhá pro všechny vstupní brány obvodu. Na řádku 2 je do d uložena budící brána k právě zpracovávané vstupní bráně. Dále je na řádcích 3 až 6 hodnota vstupní brány nastavena na stejnou hodnotu, jakou má budící brána. Podmínka rozlišuje dva případy: budící branou je výstupní brána nějakého obvodového prvku nebo budící branou je primární vstupní brána obvodu. V prvním případě se pro nastavení hodnoty vstupní brány provede řádek 4, ve druhém případě řádek 6. V obou případech je pro nastavení brány použito algoritmu *replace*. Získání identifikačního čísla vstupní brány, jež odpovídá pořadí prvku v posloupnosti VI, je zajištěno pomocí funkce *in_ord*. Pro získání aktuální hodnoty budící brány je použito funkce *get*. V případě, že budící branou je výstupní brána nějakého obvodového prvku, je přiřazovaná hodnota získána z posloupnosti VO. V tomto případě je pro získání identifikačního čísla výstupní brány, jež odpovídá pořadí prvku v posloupnosti VO, použito funkce *out_ord*. V případě, že budící branou je primární vstupní brána, je přiřazovaná hodnota získána z posloupnosti VTV. V tomto případě je pro získání identifikačního čísla primární vstupní brány, jež odpovídá pořadí prvku v posloupnosti VTV, použito funkce *pin_ord*.

6.9.3 Algoritmus funkce $pwr_1(TVS, TA, SRS, SCS, SVS, M)$

Funkce provádí zjednodušenou simulaci aplikace testu. Předpokládá se použití izolovatelných řetězců scan bez vycpávky. Funkce pracuje s testovací strategií ALAP (viz kapitola 6.1) a je použitelná i pro kombinační obvody.

Vstupem funkce je posloupnost testových cyklů TVS, posloupnost TA určující pořadí aplikace testovacích vektorů, posloupnost SRS definující topologii scan řetězců, posloupnost SCS určující pořadí zapojení registrů v řetězcích scan, posloupnost scan vektorů SVS, příkonová metrika $M \in \{NTC, WNTC, WSA\}$. Návratovou hodnotou je reálné číslo kvantifikující požadovanou příkonovou metriku.

Algoritmus $pwr_1(TVS, TA, SRS, SCS, SVS, M)$

```
1 P := 0.0
2 init_seq(VI, |IN|, 0)
3 init_seq(VO, |OUT|, INIT_V)
4 init_seq(VTV, |PI|, INIT_V)
```

```
5  update(VTV, VO, VI)
6  init_scan(SVI, SRS, INIT_V)
7  init_scan(SVO, SRS, INIT_V)
8  LSVS := SVS
9  LSCS := SCS
10 LSRS := SRS

12 Pro x := 0 až len(TA) − 1 prováděj:

14     Jestliže len(SRS) ≠ 0:
15         SIV := get(SVS, get(TA, x))

17         Pro x1 := 0 až len(SRS) − 1 prováděj:
18             Pro x2 := 0 až len(get(SRS, x1)) − 1 prováděj:
19                 replace(VO, outscan_ord(get2d(SRS, x1, get2d(SCS, x1, x2))),
20                     get2d(SIV, x1, x2))
21                 replace2d(SVI, x1, get2d(SCS, x1, x2), get2d(SIV, x1, x2))
22         P := P + pwr_scan(SVI, SVO, M)

24     VTV := get(TVS, get(TA, x))
25     update(VTV, VO, VI)

27     Pro ∀ e : e ∈ E prováděj:
28         VO := ν(e, VI, VO)
29         P = P + η(e, VI, VO, M)

31     Jestliže len(SRS) ≠ 0:
32         update(VTV, VO, VI)
33         Pro x1 := 0 až len(SRS) − 1 prováděj:
34             Pro x2 := 0 až len(get(SRS, x1)) − 1 prováděj:
35                 replace2d(SVO, x1, get2d(SCS, x1, x2), get(VI,
36                     inscan_ord(get2d(SRS, x1, get2d(SCS, x1, x2)))))

38 Jestliže len(SRS) ≠ 0:
39     init_scan(SVI, SRS, 0)
40     P := P + pwr_scan(SVI, SVO, M)
41 Vrať P
```

Popis algoritmu $pwr_1(TVS, TA, SRS, SCS, SVS, M)$**:** Na řádcích 1 až 10 je algoritmus inicializován. Do proměnné P je postupně kvantifikována příkonová metrika. V posloupnosti VI jsou udržovány hodnoty, které se vyskytují na jednotlivých vstupních branách obvodu. V posloupnosti VO jsou udržovány hodnoty, které se vyskytují na jednotlivých výstupních branách obvodu. Délka posloupností VI tedy musí být shodná s počtem prvků v množině $|IN|$ a délka posloupnosti VO musí být shodná s počtem prvků v množině $|OUT|$. Na řádku 2 je pomocí algoritmu *init_seq* posloupnost VI inicializována na délku $|IN|$ hodnotami 0. Na řádku 3 je posloupnost VO inicializována na délku $|OUT|$ inicializačními hodnotami $INIT\_V$ ($INIT\_V$ zastupuje inicializační funkci, která může vracet buď konstantní hodnotu z množiny $\{0, 1\}$, nebo náhodnou hodnotu z množiny $\{0, 1\}$, záleží na

konkrétně uvažované inicializační strategii). Posloupnost VTV udržuje hodnoty vyskytující se na jednotlivých primárních vstupech obvodu. Tedy počet prvků VTV je shodný s $|PI|$. Tato posloupnost je na řádku 4 inicializována hodnotami $INIT\_V$. Na řádku 5 jsou na všechny vstupní brány přeneseny hodnoty z odpovídajících budících bran. Na řádcích 6 a 7 jsou inicializovány dvojrozměrné posloupnosti SVI, SVO na hodnotu $INIT\_V$ pomocí algoritmu $init\_scan$. Posloupnost SVI uchovává scan vektor, jež má být sériově načten do scan registrů jednotlivých scan řetězců. Posloupnost SVO uchovává odezvu obvodu na předchozí scan vektor, jež bude sériově vyčtena při načítání SVI. Na řádcích 8 až 10 jsou posloupnosti SVS, SCS, SRS nakopírovány do pomocných proměnných, aby během činnosti algoritmu nedošlo k jejich modifikaci.

V hlavním cyklu algoritmu (řádky 12 až 36) probíhá vlastní simulace. V každém cyklu algoritmu je odsimulován jeden testový cyklus. Na řádcích 14 až 22 je kvantifikována příkonová metrika pro sériové načtení scan vektoru a sériové vyčtení odezvy obvodu (bude vysvětleno dále). Na řádku 24 je do VTV uložen aktuální testovací vektor. Při výběru testovacího vektoru je uvažováno aktuální pořadí aplikace testovacích vektorů (posloupnost TA). Na řádku 25 je testovací vektor přenesen na odpovídající vstupy obvodových prvků. Cyklus na řádcích 27 až 29 simuluje vlastní přechod všech obvodových prvků mezi jednotlivými stavy. Na řádku 28 jsou na výstupní brány nastaveny hodnoty získané aplikací přechodové funkce ν. Dále je do P přičtena příkonová metrika získaná aplikací funkce η pro daný přechod (řádek 29). Řádky 32 až 36 simulují načtení odezvy obvodu do scan řetězců. Do SVO je načtena odezva obvodu, která bude sériově vyčtena v následujícím testovém cyklu. Po odsimulování všech testových cyklů je na řádcích 38 až 40 odsimulováno sériové vyčtení poslední odezvy obvodu.

Pokud obvod obsahuje alespoň jeden scan řetězec, je na řádku 15 do proměnné SIV uložen aktuální scan vektor. Při výběru scan vektorů je uvažováno aktuální pořadí aplikace testovacích vektorů (posloupnost TA). Vnější cyklus na řádcích 17 až 21 je proveden pro všechny scan řetězce. Vnitřní cyklus na řádcích 18 až 21 je proveden pro všechny scan registry jednotlivých scan řetězců. Na řádcích 19 až 20 jsou výstupní brány jednotlivých scan registrů nastaveny na hodnoty odpovídajících bitů scan vektoru, přičemž je uvažováno aktuální pořadí zapojení scan registrů. Na řádku 21 jsou tyto bity zároveň nakopírovány do SVI. Rozdíl SVI oproti SIV je v tom, že u SVI je uvažováno aktuální pořadí zapojení jednotlivých scan registrů, kdežto u SIV nikoliv. Nyní VO reflektuje stav jednotlivých výstupních bran scan registrů po dokončení sériového scan posuvu. Protože nyní posloupnosti SVI a SVO obsahují aktuální data, tak na řádku 22 může dojít ke kvantifikování příkonové metriky pomocí algoritmu pwr_{scan}. Posloupnost SVO je během simulace na řádcích 33 až 35 aktualizována dle hodnot nacházejících se na vstupních branách odpovídajících scan registrů.

6.9.4 Algoritmus $scan\_shift(VI, VO, SIV, SV, SRS, SCS, M, P)$

Algoritmus provádí simulaci sériového posunu hodnot scan řetězci.

Vstupem je posloupnost VI uchovávající aktuální hodnoty, jež se vyskytují na vstupních branách obvodu, posloupnost VO uchovávající aktuální hodnoty, jež se vyskytují na výstupních branách obvodu, dvojrozměrná posloupnost SIV představující vstupní scan vektor, dvojrozměrná posloupnost SV představující aktuální hodnoty uložené ve scan registrech jednotlivých scan řetězců, posloupnost SRS definující topologii scan řetězců, posloupnost SCS určující pořadí zapojení registrů v řetězcích scan, posloupnost scan vektorů SVS, příkonová metrika $M \in \{NTC, WNTC, WSA\}$, proměnná P, která slouží pro kvantifikaci příkonové metriky. Výstupem jsou modifikovaná posloupnost VO, modifikovaná dvojroz-

měrná posloupnost SV a modifikovaná proměnná P.

Algoritmus $scan\_shift(VI, VO, SIV, SV, SRS, SCS, M, P)$

```
1  Pro x1 := 0 až len(car SIV) − 1 prováděj:
2      Pro x2 := 0 až len(SRS) − 1 prováděj:
3          Pro x3 := len(get(SV, x2)) − 1 dekrementuj až 1 prováděj:
4              replace2d(SV, x2, x3, get2d(SV, x2, x3 − 1))
5              P = P + ζ(get2d(SRS, x2, get2d(SCS, x2, x3)),
6                  get2d(SV, x2, x3 − 1), get2d(SV, x2, x3), M)

8          stuff_size := len(get(SIV, x2)) − len(get(SRS, x2))
9          Jestliže x1 ≥ stuff_size:
10             P := P + ζ(get2d(SRS, x2, get2d(SCS, x2, 0)),
11                 get2d(SIV, x2, get2d(SCS, x2, x1 − stuff_size)),
12                     get2d(SV, x2, 0), M)
13             replace2d(SV, x2, 0, get2d(SIV, x2,
14                 get2d(SCS, x2, x1 − stuff_size)))
15         jinak:
16             P := P + ζ(get2d(SRS, x2, get2d(SCS, x2, 0)),
17                 get2d(SIV, x2, x1), get2d(SV, x2, 0), M)
18             replace2d(SV, x2, 0, get2d(SIV, x2, x1))
19         Pro x3 := 0 až len(get(SRS, x2)) − 1 prováděj:
20             replace(VO, outscan_ord(get2d(SRS, x2, x3)), get2d(SV, x2, x3))
21     update(VTV, VO, VI)

23     Pro ∀ e : e ∈ E \ SC prováděj:
24         VO := ν(e, VI, VO)
25         P := P + η(e, VI, VO, M)
```

Popis algoritmu $scan\_shift(VI, VO, SIV, SV, SRS, SCS, M, P)$**:** Pro všechny scan řetězce, jež jsou kratší než nejdelší scan řetězec, se předpokládá, že dvojrozměrná posloupnost SIV obsahuje vycpávku tak, aby všechny posloupnosti v SIV (scan vektory pro jednotlivé scan řetězce) měly shodnou délku. Platí, že tato délka musí být shodná s $len(carSIV)$. Aktuální obsah všech scan registrů je udržován v dvojrozměrné posloupnosti SV. Vnější cyklus (na řádcích 1 až 25) proběhne pro všechny bity scan vektoru (všechny scan vektory mají shodný počet bitů). Prostřední cyklus (na řádcích 2 až 25) proběhne pro všechny scan řetězce. Na řádcích 3 až 6 je simulován posuv hodnot ve všech scan řetězcích o jednu pozici. Během posuvu je uvažováno pořadí zapojení scan registrů dle SCS. V reálném obvodu proběhne posuv všech hodnot souběžně, což je však nemožné přímo odsimulovat. Aby nedošlo k přepsání právě načítaných hodnot, je posuv simulován od posledního scan registru k prvnímu. Tedy např. pro tři scan registry ve scan řetězci nejdříve přesuneme obsah druhého registru do třetího registru a následně přesuneme obsah prvního registru do druhého registru. Posuv hodnot registry je realizován na řádku 4. Kvantifikace příkonové metriky pro posuv je provedena na řádcích 5 a 6 (vliv na připojené obvodové prvky bude započten dále). Dále je třeba do každého prvního scan registru každého scan řetězce vložit následující bit scan vektoru nebo vycpávku. Na řádku 8 je vypočtena velikost vycpávky pro daný scan řetězec. Dále se rozhodne, zda je zrovna zpracovávána vycpávka. Pokud není, jsou provedeny řádky 10 až 14. Na řádcích 10 až 12 je aktualizován obsah proměnné P. Do

prvního scan registru daného scan řetězce je vložen odpovídající bit scan vektoru (řádky 13 a 14). Pokud však zrovna je zpracovávána vycpávka, jsou provedeny řádky 16 až 18. Na řádcích 16 a 17 je aktualizován obsah proměnné P. Dále je do prvního scan registru daného scan řetězce vložena vycpávka (řádek 18). V cyklu na řádcích 19 až 20 jsou na výstupní brány scan registrů nastaveny jejich vnitřní hodnoty. Na řádku 21 je aktualizována hodnota všech vstupních bran. Cyklus na řádcích 23 až 25 probíhá pro všechny obvodové prvky. Jsou simulovány přechody prvků během posunu scan vektorů scan registry (řádek 24) a zároveň je aktualizována proměnná P (řádek 25).

6.9.5 Algoritmus funkce $pwr_2(TVS, TA, SRS, SCS, SVS, M)$

Algoritmus funkce provádí úplnou simulaci aplikace testu. Funkce pracuje s testovací strategií ALAP (viz kapitola 6.1) a je použitelná i pro kombinační obvody.

Vstupem je posloupnost testových cyklů TVS, posloupnost TA určující pořadí aplikace testovacích vektorů, posloupnost SRS definující topologii scan řetězců, posloupnost SCS určující pořadí zapojení registrů v řetězcích scan, posloupnost scan vektorů SVS, příkonová metrika $M \in \{NTC, WNTC, WSA\}$. Návratovou hodnotou je reálné číslo kvantifikující požadovanou příkonovou metriku.

Algoritmus $pwr_2(TVS, TA, SRS, SCS, SVS, M)$

```
1  P := 0.0
2  init_seq(VI, |IN|, 0)
3  init_seq(VO, |OUT|, INIT_V)
4  init_seq(VTV, |PI|, INIT_V)
5  update(VTV, VO, VI)
6  init_scan(SV, SRS, INIT_V)
7  LSVS := SVS

9  Pro x := 0 až len(TA) − 1 prováděj:

11     Jestliže len(SC) ≠ 0:
12         SIV := get(LSVS, get(TA, x))
13         scan_shift(VI, VO, SIV, SV, SRS, SCS, M, P)

15     VTV := get(TVS, get(TA, x))
16     update(VTV, VO, VI)

18     Pro ∀ e : e ∈ E prováděj:
19         VO := ν(e, VI, VO)
20         P = P + η(e, VI, VO, M)

22     Jestliže len(SRS) ≠ 0:
23         Pro x1 := 0 až len(SRS) − 1 prováděj:
24             Pro x2 := 0 až len(get(SRS, x1)) − 1 prováděj:
25                 replace2d(SV, x1, x2, get(VO, outscan_ord(get2d(SRS, x1, x2))))

27 Jestliže len(SRS) ≠ 0:
28     init_scan(SIV, SRS, 0)
```

29 $scan\_shift(VI,\ VO,\ SIV,\ SV,\ SRS,\ SCS,\ M,\ P)$
30 `Vrať` P

Popis algoritmu $pwr_2(TVS, TA, SRS, SCS, SVS, M)$**:** Na řádcích 1 až 7 je algoritmus inicializován podobným způsobem jako dříve popsaný algoritmus pwr_1. Rozdíl je v tom, že místo dvojrozměrných posloupností SVI a SVO, které uchovávaly vstupní a výstupní scan vektor, je v tomto případě použita dvojrozměrná posloupnost SV. V této posloupnosti je uchováván aktuálního obsah všech scan registrů. Srdcem algoritmu je smyčka na řádcích 9 až 25, která probíhá pro všechny testové cykly. Pokud obvod obsahuje alespoň jeden scan řetězec, jsou provedeny řádky 12 až 13. Nejdříve jsou pro daný testový cyklus získány odpovídající vstupní scan vektory pro jednotlivé scan řetězce, přičemž je uvažováno aktuální pořadí aplikace testovacích vektorů. Tyto scan vektory jsou uloženy do dvojrozměrné posloupnosti SIV (řádek 12) a dále jsou načteny do scan řetězců pomocí aplikace algoritmu $scan\_shift$ (řádek 13). Na řádku 15 je získán testovací vektor, který bude nastaven na primární vstupy obvodu, přičemž je uvažováno aktuální pořadí aplikace testovacích vektorů. Na řádku 16 jsou hodnoty z primárních vstupů přeneseny na vstupní brány jednotlivých obvodových prvků. Ve smyčce na řádcích 18 až 20 jsou aplikovány testovací vektory na obvod. V této smyčce je aplikována přechodová funkce ν na všechny obvodové prvky (řádek 19) a zároveň je aktualizována proměnná P (řádek 20). Řádky 23 až 25 se provedou, pokud obvod obsahuje alespoň jeden scan řetězec. Na těchto řádcích je simulováno sejmutí odezvy obvodu pomocí scan registrů. Vnější smyčka na řádcích 23 až 25 probíhá pro všech scan řetězce. Vnitřní smyčka na řádcích 24 až 25 probíhá pro všechny scan registry jednotlivých scan řetězců. Do každého scan registru je načtena hodnota z jeho vstupní brány (řádek 25). Pokud obvod obsahuje alespoň jeden scan řetězec, tak je po odsimulování všech testových cyklů ještě potřeba sériově vyčíst ze všech scan řetězců odezvu obvodu na poslední testovací vektor, což je zajištěno na řádcích 27 až 29.

6.10 Algoritmy pro ohodnocení a výběr jedinců

V této kapitole bude definován algoritmus *phi* pro kvantitativní ohodnocení kvality jedinců populace (funkce fitness). Algoritmus převádí genotyp na fenotyp. Fenotyp je představován posloupností, jež definuje pořadí aplikace testovacích vektorů a pořadí zapojení registrů do jednotlivých řetězců scan. Toto pořadí je uvažováno během simulace aplikace testu, kdy dochází ke kvantifikování příkonové metriky. Výsledná hodnota je dále přepočítána na hodnotu fitness tak, aby velikost hodnoty fitness byla přímo úměrná kvalitě jedince. Jedinec bude tím kvalitnější, čím vyšší bude jeho hodnota fitness. Hodnota fitness jedince bude tím vyšší, čím nižší příkon bude dosažen během aplikace testu.

Dále budou v této kapitole popsány dva nejrozšířenější způsoby výběru jedinců z populace pro aplikaci operátoru křížení. Bude popsán způsob výběru jedinců pomocí kola štěstí a způsob výběru jedinců pomocí turnaje. Způsob výběru jedinců pomocí turnaje bude prezentován pomocí algoritmu.

6.10.1 Algoritmus $phi(TVS, SRS, SVS, CH, K, M)$

Algoritmus realizuje vyhodnocení funkce Φ (fitness) pro genotyp daného jedince populace (chromozóm).

Vstupem je posloupnost testovacích vektorů TVS, posloupnost identifikátorů scan registrů SRS, scan vektory SVS, chromozóm CH, posloupnost K popisující rozložení bloků

v chromozómu, metrika použitá pro stanovení příkonu $M \in \{NTC, WNTC, WSA\}$. Návratovou hodnotou je reálné číslo určující kvalitu (fitness) chromozómu CH.

Algoritmus $phi(TVS, SRS, CH, K, M)$

1 $delta(CH,\ K,\ TA,\ SCS)$
2 **Vrať** 1 / $pwr_2(TVS,\ TA,\ SRS,\ SCS,\ SVS,\ M)$

Popis algoritmu $phi(TVS, SRS, SVS, CH, K, M)$**:** Nejdříve je na řádku 1 pomocí aplikace algoritmu *delta* převeden chromozóm CH na posloupnosti TA a SCS. Tyto posloupnosti jsou využity během následné simulace aplikace testu. Pro simulaci aplikace testu je možné použit buď rychlejší a méně přesný algoritmus pwr_1 nebo pomalejší a přesnější algoritmus pwr_2 (detaily viz kapitola 6.9). V definici je implicitně uvažován algoritmus pwr_2. Oba algoritmy během simulace kvantifikují požadovanou příkonovou metriku. Hodnota fitness je vypočtena jako převrácená hodnota z hodnoty získané simulací. Takto stanovená hodnota fitness je vrácena volajícímu programu. Všechny tyto kroky jsou realizovány na řádku 2.

6.10.2 Výběr jedinců pomocí operátoru s

Operátor s vybírá jedince pro aplikaci operátoru křížení. Existují různé způsoby výběru jedinců. Zpravidla jsou při výběru upřednostňováni jedinci s vyšší hodnotou fitness, nicméně je vhodné zajistit, aby i jedinci s nižší hodnotou fitness měli šanci při výběru uspět, protože jejich genetická informace může vést k nalezení kvalitních řešeních v budoucích generacích. Většinou jsou jedinci vybírání tak, aby se z nich daly utvořit dvojice, jež budou vzájemně kříženy. Křížením dvojice vznikají potomci, jež přecházejí do nové generace a rodiče zanikají (v případě, že není uplatňován elitismus). V praxi se nejčastěji využívá výběru jedinců pomocí kola štěstí (ruleta) nebo turnaje.

6.10.2.1 Výběr jedinců pomocí kola štěstí

V případě kola štěstí je jedinec i vybrán s pravděpodobností $p_{CH_i} \in \langle 0; 1 \rangle$ dle rovnice (6.1).

$$p_{CH_i} = \frac{phi(TVS, SRS, CH_i, K)}{\sum_{j=1}^{\lambda} phi(TVS, SRS,\ CH_j, K)} \tag{6.1}$$

Výběr jedinců lze v tomto případě přirovnat ke hře ruleta, kdy jednotlivá pole rulety zastupují jedince populace a velikost polí rulety odpovídá pravděpodobnosti výběru jedince dle rovnice (6.1).

6.10.2.2 Algoritmus pro výběr jedinců pomocí turnaje

V případě turnaje je vybráno k_1 jedinců, kteří mezi sebou sehrají turnaj. Výsledkem je výběr k_2 jedinců, přičemž se uplatňuje přednastavená pravděpodobnost p.

Vstupem jsou konstanty k_1, k_2 ($k_1 \geq k_2$) a pravděpodobnost p, které ovlivňují výběr. Výstupem je množina vybraných prvků O, $|O| = k_2$.

Algoritmus pro výběr jedinců pomocí turnaje

1 $A := \{\}$
2 $O := \{\}$
3 $i := k_2$

```
4  q := 1 − p
5  Vyber k1 jedinců z populace náhodným způsobem a vlož je do množiny A
6  Dokud i ≠ 0 prováděj:
7      B := A
8      Dokud |B| ≠ 0 ∨ i ≠ 0 prováděj:
9          Vyber a, a ∈ B s největším Φ
10         S pravděpodobností p proveď:
11             O := O ∪ {a}
12             A := A \ {a}
13             i := i − 1
14         p := p ∗ q
15         q := q ∗ (1 − p)
16         B := B \ {a}
```

Popis algoritmu turnaje: Algoritmus je inicializován na řádcích 1 až 4. Na řádku 5 je náhodně vybráno k_1 jedinců z populace do množiny A. Cyklus na řádcích 6 až 16 realizuje vlastní turnaj mezi k_1 jedinci, jehož výsledkem je volba k_2 jedinců. Cyklus probíhá dokud není vybráno všech k_2 jedinců. Na řádku 7 je pomocná proměnná B naplněna jedinci z A. Vnitřní cyklus na řádcích 8 až 16 zajišťuje zpracování všech jedinců z B nebo předčasné ukončení v případě vybrání potřebného počtu jedinců. Na řádku 9 je vybrán z B jedinec s největší hodnotou fitness, který je dále s pravděpodobností p zpracován (řádky 10 až 13). Pokud je jedinec zpracován, je přidán do množiny O (řádek 11). Dále je jedinec odstraněn z množiny A (řádek 12), aby nebyl opětovně zpracováván. Potom je dekrementován počet jedinců zbývajících ještě k vybrání (řádek 13). Po každém průchodu vnitřním cyklem je upravena pravděpodobnost výběru dalšího jedince (řádky 14 a 15) a jedinec je odstraněn z pomocné množiny B (řádek 16). Po vyčerpání všech jedinců z B, pokud stále ještě nebylo vybráno všech k_2 jedinců, dochází k opětovnému naplnění B dosud nevybranými jedinci z A (řádek 7).

6.11 Způsob vytvoření prvotní populace

Před aplikací GA je třeba vytvořit prvotní populaci jedinců. Prvotní populaci je možné vygenerovat náhodně. Vhodnější však je jako základ pro vytvoření prvotní populace použit výstup z profesionálního návrhového systému. Návrhový systém generuje testovací vektory i propojení registrů scan v určitém pořadí, které však není optimální z hlediska příkonu. Z takto získaných posloupností lze vytvořit chromozóm pomocí aplikace algoritmu *delta_inv*. Jedinec s tímto chromozómem se dále vloží do prvotní populace. V případě, že je použit elitismus, je takto zajištěno, že nejlepší řešení nalezené pomocí GA nebude nikdy horší než řešení navržené návrhovým systémem. Zbytek populace lze doplnit náhodně.

Kapitola 7

Implementace metodiky

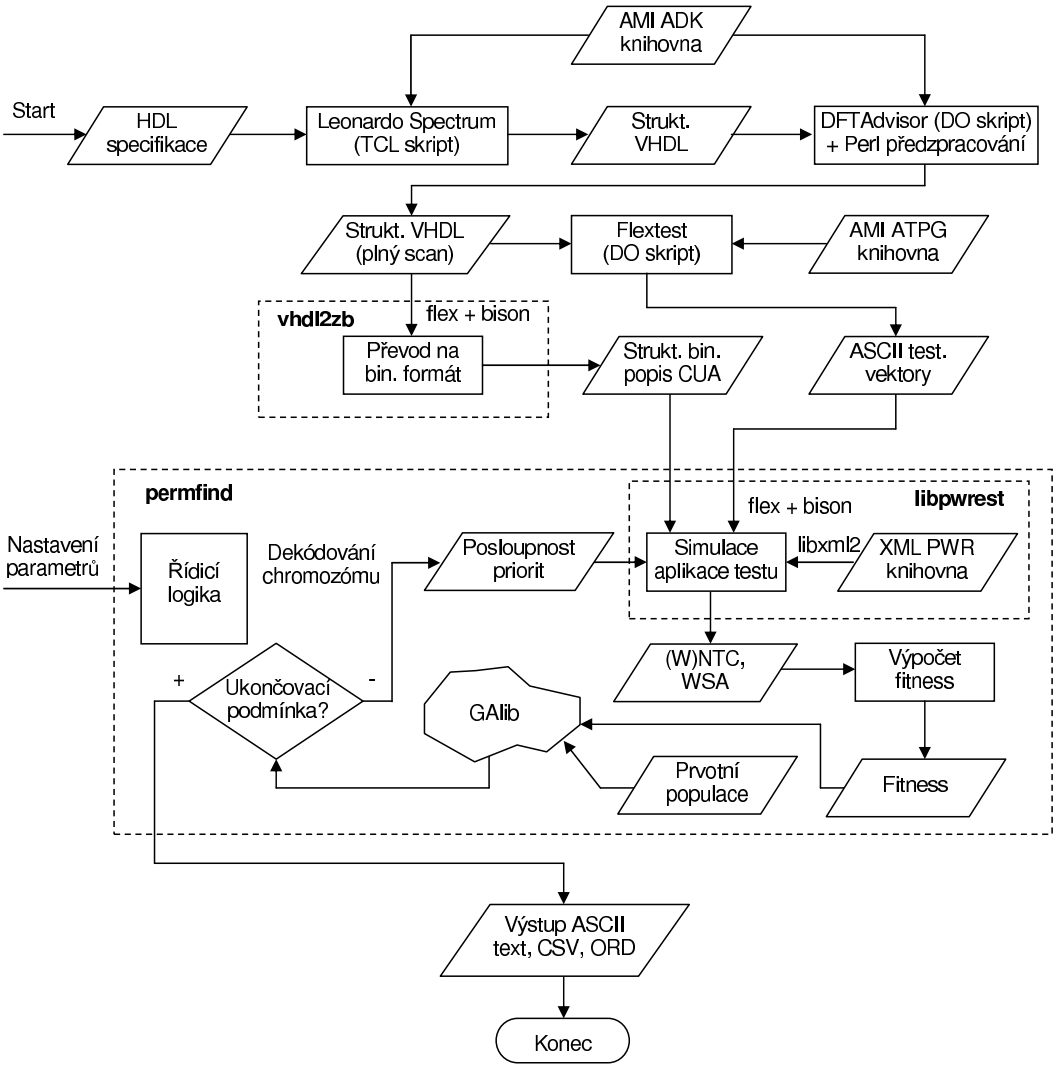
Implementace metodiky (včetně dokumentace) je zveřejněna na URL [76]. V rámci práce byl vytvořen přenositelný C++ kód, který je přeložitelný pro nejrozšířenější platformy. Implementace využívá knihovny STL, takže je možné pracovat i s velice rozsáhlými číslicovými obvody. Maximální počet prvků číslicového obvodu je tak prakticky omezen velikostí dostupné operační paměti. Implementace byla uvolněna pod licencí GNU GPL verze 3 nebo vyšší (uživatel má možnost zvolit si verzi licence, která mu nejvíce vyhovuje). Aplikace se sestavuje pomocí systému GNU Autotools. Pro překlad se doporučuje použít překladač GCC. Překlad byl úspěšně realizován na OS Linux (překladač GCC) i Windows (překladače Cygwin GCC a MinGW) pro 32 bitovou i 64 bitovou architekturu.

7.1 Popis implementace

Na obr. 7.1 lze vidět zjednodušené blokové schéma implementace, ve kterém jsou vyznačeny použité technologie a použité souborové formáty. Kosodélníky označují jednotlivá vstupní a výstupní data. Každý obdélník představuje proces zpracování dat. Proces zpracování je spuštěn, pokud jsou k dispozici všechna vstupní data. Výsledkem zpracování jsou výstupní data. Obdélníky zakreslené přerušovanou čarou vyznačují implementované softwarové aplikace a implementované softwarové knihovny.

Vstupem je specifikace CUA v HDL (VHDL nebo Verilog). Nejdříve dochází k předzpracování pomocí profesionálních nástrojů Mentor Graphics. Pro automatizaci jednotlivých kroků předzpracování byly vytvořeny skripty v jazycích TCL, Perl a DO skripty pro příkazové interpretry nástrojů DFTAdvisor a Flextest. Prezentovaný způsob předzpracování byl použit pro experimenty, jejichž výsledky budou prezentovány v kapitole 8. Teoreticky je možné použít i jiný způsob předzpracování využívající jiných nástrojů. V takovém případě se však bude postup předzpracování pravděpodobně lišit od dále popisovaného.

HDL zápis je nejdříve syntetizován pomocí nástroje Leonardo Spectrum. Syntéza může být provedena např. pomocí vytvořeného TCL skriptu, protože nástroj Leonardo Spectrum obsahuje interpretr jazyka TCL. V tomto kroku dochází k namapování CUA na prvky knihovny AMI, čímž vznikne strukturální VHDL. Pro další předzpracování je využit skript v jazyce Perl. V tomto kroku jsou přejmenovány názvy obvodových prvků a bran tak, aby neobsahovaly speciální „escape“ sekvence. Tyto sekvence používá nástroj Leonardo Spectrum pro identifikaci automaticky vkládaných signálů a obvodových prvků. Tyto sekvence mohou být bezpečně odstraněny, aniž by došlo ke změně funkce obvodu. Názvy prvků a bran, jež jsou delší než dvacet znaků, jsou při předzpracování zkráceny na unikátní názvy



Obrázek 7.1: Implementace

s délkou do dvaceti znaků. Vícebitové sběrnice jsou během předzpracování rozvinuty na jednobitové signály.

V případě, že se nejedná o kombinační obvod, je pomocí nástroje DFTAdvisor do návrhu vložen plný řetězec scan. Pro automatizaci tohoto kroku je možné použít vytvořený DO skript, jež obsahuje příkazy pro příkazový interpretr nástroje DFTAdvisor. Dále jsou pomocí nástroje Flextest vygenerovány testovací vektory, k čemuž lze využít další DO skript, jež byl vytvořen pro příkazový interpretr nástroje Flextest. Vzniká ASCII soubor obsahující jednotlivé vygenerované testovací vektory. Strukturální VHDL s vloženým řetězcem scan je dále pomocí implementované aplikace *vhdl2zb* převedeno na binární formát (detailní popis tohoto formátu viz [76]). Díky tomu je zjednodušeno další zpracování, protože pak již není třeba pracovat s komplexním VHDL kódem.

Strukturální popis obvodu v binárním formátu a vygenerované testovací vektory jsou vstupem implementované aplikace *permfind*, která využívá GA pro optimalizaci testu pro nízký příkon. Činnost GA vyžaduje simulaci aplikace testu. Simulace aplikace testu byla implementována v samostatné objektové knihovně *libpwrest*, takže tento simulátor může být dále využit i v jiných projektech. Vytvořená aplikace *permfind* čte parametry z příkazové řádky. Uživatel si zvolí požadovanou příkonovou metriku, která bude kvantifikována během simulace, nastaví parametry GA, vybere binární soubor popisující CUA a soubor testovacích vektorů. Pokud soubor testovacích vektorů není kompatibilní (viz kapitola 6.1) , pokusí se jej aplikace automaticky upravit s využitím postupů popsaných v kapitole 6.2. Pokud nelze úpravu provést, je vypsána chyba a další zpracování je ukončeno. Výstupem aplikace je textový soubor s navrženým pořadím aplikace testovacích vektorů a s navrženým pořadím zapojení registrů do řetězců scan (v případě plného scanu). Uživatel je též informován o hodnotě příkonových metrik zjištěných simulací pro vstupní obvod a pro obvod po aplikaci navržené optimalizace. Dosažení redukce příkonu je uživateli oznámeno.

Aplikace dále umožňuje zvolit režim procházení stavového prostoru úlohy. Je možné zvolit procházení celého stavového prostoru (tzv. brutální síla) nebo procházení pomocí GA ukončujícího po dosažení stanovené konvergence nebo procházení pomocí GA ukončujícího po stanoveném počtu generací. Parametry pro jednotlivé režimy mohou být nastaveny ručně nebo automaticky. Jednotlivé režimy jsou blíže popsány v kapitole 7.2. Aplikace dále podporuje simulační režim, ve kterém je možné odsimulovat aplikaci testu pro přednastavená vstupní data bez optimalizace. Pro vyhodnocení příkonu jsou podporovány metriky NTC, $WNTC$, WSA a Hammingova vzdálenost mezi testovacími vektory. Je možné zvolit samostatnou metriku pro optimalizaci a samostatnou metriku pro závěrečnou simulaci, díky čemuž lze např. zkoumat vztahy mezi jednotlivými metrikami. V případě použití GA lze nastavit počet samostatných běhů GA. Výsledky z jednotlivých běhů mohou být průměrovány nebo mohou být uchovávány pouze nejlepší dosažené hodnoty.

Aplikace měří délku trvání běhu s rozlišením na milisekundy. Pro GA lze nastavit velikost populace, počet generací, pravděpodobnost aplikace operátoru mutace, pravděpodobnost aplikace operátoru křížení, cílovou konvergenci a počet generací, přes které je počítána konvergence. Aplikace umožňuje vytvářet soubory obsahující statistická data pro jednotlivé generace GA. Tyto statistiky pro každou generaci uchovávají minimální, maximální, průměrnou hodnotu fitness, střední odchylku fitness a číslo kvantifikující diversitu populace. Dále je možné zvolit, zda budou během simulace zvlášť kvantifikovány příkonové metriky pro aplikaci testovacích vektorů a zvlášť pro scan řetězce nebo zda budou kvantifikovány dohromady. Způsob inicializace jednotlivých obvodových bran je volitelný. Je možné zvolit zda budou jednotlivé brány inicializovány na hodnotu „logická 0“, „logická 1“ nebo náhodně. Taktéž je možné zvolit způsob zpracování „don’t care“ bitů v testovacích vektorech.

Tyto bity mohou být během simulace nastaveny na hodnotu „logická 0“, „logická 1“ nebo na náhodnou hodnotu. Obdobným způsobem lze zvolit způsob způsob chování v případě nepřipojených vstupů. Na těchto vstupech mohou být během simulace uvažovány hodnoty „logická 0“, „logická 1“ nebo náhodné hodnoty. Dále lze zvolit strmost hran vstupních signálů, jež bude uvažována během simulace.

Pro vstup testovacích vektorů je podporován formát Flextest ASCII a zjednodušený textový formát (tzv. RAW) obsahující textově zapsanou posloupnost aplikace jednotlivých testovacích vektorů (detaily viz [76]). Nalezené pořadí aplikace testovacích vektorů a zapojení registrů do řetězce scan je možné uložit do tzv. ORD souboru. Pro samostatný režim simulace je potom kdykoliv možné použít libovolný tento ORD soubor a ověřit dříve získané výsledky např. s jiným nastavením simulátoru. Veškeré dosažené výsledky mohou být uživateli prezentovány v textové podobě anebo automaticky formátovány do CSV souboru, který je pak možné snadno pracovávat dalšími nástroji (např. tabulkový procesor, Gnuplot, atp.).

Časově nejnáročnějším krokem je ohodnocení kvality jedinců v populaci, protože pro každého jedince v populaci je třeba provést simulaci aplikace testu. Toto ohodnocení může probíhat paralelně. Implementace proto podporuje v dnešní době již velice rozšířené multiprocesorové systémy. Automaticky se detekuje počet dostupných CPU a ohodnocení kvality jedinců se rozděluje mezi dostupné CPU. Uživatel může omezit počet CPU, které budou použity. Rozdělování činnosti mezi CPU probíhá dynamicky po jednotlivých generacích GA, takže volným CPU je přidělováno více práce než CPU vytíženým jinými úlohami. Vytížené CPU tedy zbytečně nezpomalují běh celé úlohy. Pokud je k dispozici více CPU než je požadovaný počet jedinců v populaci, bude použito jen tolik CPU, kolik je jedinců v populaci. V tomto případě jedinou možností jak využít všechny dostupné CPU, je ručně zvýšit počet jedinců v populaci. Vzhledem k tomu, že populace běžně obsahuje stovky jedinců a CPU na multiprocesorových systémech jsou v dnešní době k dispozici jednotky, nemělo by toto být výraznějším omezením. Pro plné využití multiprocesorových systémů je však nutné, aby překladač podporoval technologii OpenMP. U překladače GCC je technologie OpenMP podporována od verze 4.2. V případě překladu pomocí nižší verze GCC bude i u multiprocesorových systémů výpočet probíhat pouze na jednom procesoru.

Při implementaci bylo snahou ošetřit většinu možných chyb a problémů s konzistencí jednotlivých souborů, jež se mohou při používání aplikace vyskytnout. O každém problému, který nastane při běhu aplikace, je uživatel přehledně informován. V případě fatálních problémů je běh aplikace ukončen a operačnímu systému je vrácen kód specifikující danou chybu, což umožňuje snadné nasazení aplikace ve skriptech. Vytvořená implementace obsahuje vestavěnou nápovědu a během instalace jsou na cílový systém kopírovány i manuálové stránky. Na URL [76] jsou k dispozici ke stažení spustitelné binární soubory pro Windows i Linux, dále soubory se zdrojovým kódem, RPM i DEB balíčky a ebuild soubory pro instalaci na nejrozšířenější Linuxové distribuce.

7.2 Režimy činnosti

Pro procházení stavového prostoru úlohy implementace umožňuje zvolit jeden z pěti režimů činnosti:

Procházení celého stavového prostoru úlohy – Tento režim je použitelný jen pro velice jednoduché obvody vzhledem k rozsáhlosti stavového prostoru úlohy. V tomto režimu je vždy identifikováno nejlepší řešení ze stavového prostoru úlohy. Tento režim

byl implementován pro ověření metodiky. Tento režim bude dále označován zkratkou BS (Brutální Síla).

Plně automatický režim ukončující počtem generací – Tento režim je přednastaven jako implicitní. V tomto režimu je pro procházení stavového prostoru použit GA, který je ukončen po odsimulování stanoveného počtu generací. Parametry GA jsou nastaveny automaticky dle vlastností vstupního obvodu. Algoritmus pro automatické nastavení parametrů je empirický a jeho cílem je dosáhnout kvalitních výsledků v krátkém čase bez nutnosti zásahu uživatele. Přestože jsou tyto požadavky vzájemně protichůdné, algoritmus se osvědčil a produkuje použitelné výsledky. Přednastavené parametry jsou zobrazeny uživateli, který je může použít jako základ pro nastavení manuálního režimu. Algoritmus pro automatické přednastavení parametrů GA je ve vývoji. Cílem je umožnit uživateli dosáhnout kvalitních výsledků, aniž by se musel zabývat jednotlivými parametry GA. Verze algoritmu implementovaná před odevzdáním této práce pracovala s následující rovnicí: $n_{gen} = 10(len(TVS)+|SC|)$, tedy počet simulovaných generací byl vypočten jako součet počtu testovacích cyklů a počtu scan registrů vynásobený deseti. Velikost populace λ byla přednastavena na 100. Pravděpodobnost operátoru mutace a pravděpodobnost operátoru křížení byla přednastavena následovně: $p_m = 0,001$, $p_c = 0,9$. Tento režim bude dále označován zkratkou AG (Automatický režim ukončující počtem Generací).

Manuální režim ukončující počtem generací – GA je ukončen po odsimulování stanoveného počtu generací, uživatel nastavuje parametry GA ručně. Tento režim bude dále označován zkratkou MG (Manuální režim ukončující počtem Generací).

Automatický režim ukončující konvergencí – GA je ukončen, jakmile je detekován pokles rychlosti konvergence. Ukončující podmínka je nastavena automaticky a parametry jsou zobrazeny uživateli, který je může použít při nastavování manuálního režimu. Konvergence je vyčíslována dle rovnice: $k(i,j) = \frac{r_{best}(i-j_{set})}{r_{best}(i)}$, kde $r_{best}(i)$ je nejvyšší dosažená redukce v generaci i a $r_{best}(i-j_{set})$ je nejvyšší dosažená redukce v generaci $i-j_{set}$, i je aktuální generace, pro kterou je konvergence počítána a j_{set} je počet generací, přes které je konvergence sledována. GA je ukončen, pokud $k(i) \geq k_{set}$, kde k_{set} je automaticky stanovený parametr. Ukončující podmínka začne být sledována až po odsimulování j_{set} generací. Ve verzi algoritmu implementované před odevzdáním této práce bylo přednastaveno $k_{set} = 0,99$, $j_{set} = 50$. Ostatní parametry byly přednastaveny stejně jako u režimu AG. Tento režim bude dále označován zkratkou AK (Automatický režim ukončující Konvergencí).

Manuální režim ukončující konvergencí (MK) – GA je ukončen, jakmile je detekován pokles rychlosti konvergence. Ukončující podmínka je shodná s režimem AK, parametry jsou nastavovány uživatelem. Tento režim bude dále označován zkratkou MK (Manuální režim ukončující Konvergencí)

7.3 Implementační detaily

Pro zpracování strukturálního VHDL kódu a testovacích vektorů ve formátu Flextest ASCII byly implementovány lexikální analyzátory s využitím nástroje GNU Flex. Pro jednotlivé formáty byly definovány gramatiky, pro které byly následně implementovány syntaktické analyzátory s využitím nástroje GNU Bison.

Pro zápis technologické knihovny byl zvolen formát XML. Bylo navrženo XML schéma, jež definuje strukturu knihovny. XML schéma bylo specifikováno v jazyce RELAX NG a je dostupné na URL projektu [76]. Díky tomu lze do projektu snadno přidávat další knihovny. Pro zpracování XML implementace využívá „open source“ knihovny libxml2. Je použito průběžné syntaktické analýzy, takže mohou být zpracovávány i XML knihovny větší velikosti, než je velikost dostupné operační paměti. Použitá XML knihovna je během načítání průběžně validována. Pokud není knihovna validní, je načítání ukončeno a uživatel je informován o příčině problému. V rámci řešení práce byly vytvořeny knihovny pro technologie AMI $0,5\,\mu m$ a AMI $1,2\,\mu m$, každá ve variantě *slow*, *typ*, *fast*. XML knihovny pro technologii AMI byly sestaveny z informací poskytovaných firmou Mentor Graphics. Pro automatizované sestavení knihoven z těchto informací byl použit program v jazyce Perl.

Pro implementaci GA byla použita „open source“ knihovna GAlib. Implementace algoritmu 6.8.2 (*delta_inv*) a algoritmu 6.10.1 (*phi*) se trochu odlišuje od původních definic. Pro zvýšení diverzity je při generování chromozómu pomocí algoritmu *delta_inv* zvoleno číslo NUM tak, aby platilo $NUM \geq 1 \wedge l \times NUM \leq max\ B$ (l je délka chromozómu, $max\ B$ je nejvyšší možná hodnota, kterou mohou nabývat jednotlivé geny). Na řádku 14 algoritmu 6.8.2 je pak do chromozómu CH místo původní hodnoty i vkládána hodnota $i \times NUM$. V implementaci publikované před odevzdáním této práce byla hodnota NUM přednastavena na hodnotu 10. Jak bylo zjištěno experimenty, vliv aktuálně použitá hodnoty NUM na kvalitu dosažených výsledků je zanedbatelný. Příklad 7.3.1 prezentuje případ pro $NUM = 4$ pro stejné vstupy jako byly použity v příkladu 6.8.2, takže je možné porovnat rozdíl ve vygenerovaném chromozómu.

Příklad 7.3.1:
$TA = (1, 2, 3)$
$SCS = ((1, 2, 3), (1, 2))$

$delta\_inv(TA, SCS, CH, K)$
$CH = (0, 4, 8, 12, 16, 20, 24, 28)$
$K = (3, 6)$

Knihovna GAlib uchovává hodnoty fitness v datovém typu Float, což při použití algoritmu 6.10.1 zanášelo nepřesnosti způsobené nedostatečným rozsahem datového typu Float a zaokrouhlovací chyby. Z tohoto důvodu bylo při implementaci použito rovnice (7.1).

$$\Phi = \frac{PWR_{orig}}{pwr_2(TVS, TA, SRS, SCS, SVS, M)} \tag{7.1}$$

V rovnici (7.1) hodnota PWR_{orig} je stanovena během prvotní simulace, pro kterou je použito pořadí prvků tak, jak bylo navržené návrhovým systémem. Jedinou nevýhodou tohoto přístupu, je nutnost provést prvotní simulaci. Prodloužení délky běhu je však vzhledem k počtu simulací prováděných během činnosti GA zanedbatelné.

Kapitola 8

Dosažené výsledky

Pro experimenty byly použity obvody z benchmarkových sad ISCAS85 [1], ISCAS89 [2], ITC99 [3]. Tyto obvody byly namapovány na technologickou knihovnu AMI 0.5 μm pomocí nástroje Leonardo Spectrum. U kombinačních obvodů byla realizována pouze optimalizace pořadí aplikace testovacích vektorů. Sekvenční obvody byly upraveny na plný scan pomocí nástroje DFTAdvisor, přičemž byl použit pouze jeden řetězec scan. U sekvenčních obvodů byla realizována jak optimalizace pořadí aplikace testovacích vektorů tak i optimalizace pořadí zapojení registrů do řetězce scan. Testovací vektory byly vygenerovány pomocí nástroje Flextest. Pro generování testu byl použit chybový model uváznutí na hodnotě logická „0“, logická „1“.

Všechny experimenty byly provedeny pomocí implementovaných programů. Tyto programy umožňují uživateli nastavit desítky různých parametrů. Detailní popis všech parametrů lze nalézt v uživatelském manuálu, jež je distribuován společně s programy. Je možné, že dostatečnou trpělivostí při nastavování jednotlivých parametrů se podaří dosáhnout i lepších výsledků, než které jsou prezentovány v této práci. Implementované programy včetně zdrojových kódů, dokumentace a popisu formátů potřebných pro sestavení vlastní knihovny (v případě, že je třeba použít jinou knihovnu než AMI), je k dispozici na URL projektu [76].

V kapitole 8.1 jsou charakterizovány obvody použité pro ověření metodiky. Kapitola 8.2 porovnává GA s procházením celého stavového prostoru úlohy z hlediska potřebného času a kvality dosažených výsledků. Kapitola 8.3 ověřuje vliv jednotlivých parametrů GA na kvalitu nalezeného řešení a rychlost konvergence GA. V kapitole 8.4 jsou zkoumány vztahy mezi jednotlivými příkonovými metrikami. Kapitola 8.5 porovnává výsledky dosažené sekvenční a souběžnou optimalizací pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězců scan. V kapitole 8.6 je provedeno porovnání s existujícími metodami. V kapitole 8.7 jsou prezentovány další výsledky dosažené s benchmarkovou sadou ITC99. Kapitola 8.8 se zabývá škálovatelností úlohy.

8.1 Charakteristika obvodů použitých pro ověření metodiky

Většinu benchmarkových obvodů se podařilo získat ve formátu Verilog nebo VHDL. Obvody specifikované popisem chování bylo možné většinou přímo syntetizovat pomocí nástroje Leonardo Spectrum. Některé obvody specifikované strukturálním popisem však bylo nutné nejdříve upravit na syntetizovatelnou formu. Problémy se vyskytovaly, pokud byly v návrhu použity nestandardní obvodové prvky, ke kterým nebyl k dispozici popis chování. Tyto

Tabulka 8.1: Vlastnosti vybraných obvodů z benchmarkové sady ISCAS.

Označení obvodu	funkce
c880	8 bitová ALU
c1355	32 bitový SEC
c1908	16 bitový SEC/DED
c2670	12 bitová ALU s integrovaným řadičem
c3540	8 bitová ALU
c5315	9 bitová ALU
c6288	16×16 násobička
c7552	32 bitový komparátor/sčítačka
s298	řadič dopravní signalizace (semafor)
s344	4×4 sčítačka/posuvný registr/násobička
s349	4×4 sčítačka/posuvný registr/násobička

prvky byly ručně nahrazeny prvky standardní obvodové knihovny. Vhodná náhrada byla zvolena dle názvu původního prvku (např. u prvků, jejichž název začínal ADD bylo možné předpokládat, že se jedná o sčítačky) a dle kontextu použití prvku, pokud byla známa funkce obvodu.

Obvody z benchmarkové sady ISCAS, jejichž název začíná písmenem „c“, jsou čistě kombinační obvody. Obvody, jejichž název začíná písmenem „s“, jsou sekvenční obvody. Pravděpodobně díky tomu, že tyto obvody byly původně distribuovány ve formě seznamu spojů, nebyly k dispozici jejich modely na vyšší úrovni abstrakce. Z tohoto důvodu je problematické specifikovat jejich funkci. V práci M. Hansena a kolektivu [26] byla s využitím metod reverzního inženýrství zjištěna funkce některých těchto obvodů. Obvody ISCAS, ke kterým se podařilo zjistit funkci, jsou uvedeny v tabulce 8.1. U těchto obvodů je velice pravděpodobné, že syntézou a přemapováním na prvky z knihovny AMI nedošlo ke změně jejich funkce. Ostatní obvody ISCAS byly přemapovány na technologickou knihovnu AMI bez znalosti jejich funkce a nelze tudíž úplně vyloučit drobné odchylky od jejich původní funkce.

Benchmarková sada ITC99 [3] obsahuje složitější sekvenční obvody, jež mají definovanou funkci. Funkce vybraných obvodů z této benchmarkové sady jsou uvedeny v tabulce 8.2. V experimentech bylo použito druhé vydání ITC99 (ITC99 second edition).

U všech těchto obvodů je vyloučeno procházení celého stavového prostoru úlohy, protože by bylo časově neúnosné. Z tohoto důvodu byl pro účely dalších experimentů náhodným způsobem vygenerován jednoduchý syntetický obvod, u nějž je možné projít celý stavový prostor úlohy a posoudit úspěšnost navržené metody.

8.2 Porovnání GA s procházením celého stavového prostoru úlohy

Tyto experimenty byly provedeny pro porovnání kvality řešení získaných pomocí GA s procházením celého stavového prostoru úlohy (režim BS, viz kapitola 7.2). Při procházení celého stavového prostoru lze vždy nalézt nejlepší možné řešení, nicméně vzhledem ke složitosti problému je tento způsob řešení časově neúnosný již pro malé obvody, protože dle kapitoly 4.3.1: $O(n) = n!$, kde n je počet prvků, jejichž pořadí má být optimalizováno.

Tabulka 8.2: Vlastnosti vybraných obvodů z benchmarkové sady ITC99.

Označení obvodu	funkce
b01	konečný automat pro porovnávání dat
b02	konečný automat pro rozpoznávání BCD čísel
b03	arbitr zdrojů
b04	výpočet minimální a maximální hodnoty
b05	šíření obsahu paměti
b06	řadič přerušení
b07	počítání bodů na rovné lince
b08	hledání podmnožin v sekvencích čísel
b09	převodník sériových dat
b10	volební systém
b11	kryptování řetězce
b12	hra pro jednoho hráče (hádání pořadí)
b13	rozhraní k meteo senzorům
b14	procesor Viper (podmnožina)
b15	procesor 80386 (podmnožina)

Tabulka 8.3: Velikost stavového prostoru úlohy dle typu použité optimalizace pro obvod b15.

Optimalizace	**Velikost stavového prostoru**		
	Rovnice	**Dosazení**	**Vyčíslení**
Pořadí testovacích vektorů	$len(TVS)!$	1297!	$1,44.10^{3476}$
Pořadí scan registrů	$\|SC\|!$	416!	$3,84.10^{910}$
Obě optimalizace sekvenčně	$len(TVS)! + \|SC\|!$	1297! + 416!	$1,44.10^{3476}$
Obě optimalizace současně	$len(TVS)!.\|SC\|!$	1297!.416!	$5,54.10^{4386}$

V tabulce 8.3 lze vidět velikost stavového prostoru úlohy pro obvod b15 při použití různých způsobů optimalizace. V prvním sloupci tabulky je uveden typ optimalizace. Ve druhém sloupci je uvedena obecná rovnice pro výpočet velikosti stavového prostoru (význam symbolů byl vysvětlen v kapitole 6). Ve třetím sloupci je provedeno dosazení do rovnice a v posledním sloupci tabulky je uvedena vyčíslená velikost stavového prostoru. Z tabulky 8.3 je zřejmé, že úloha současné optimalizace má největší stavový prostor.

Pro následující experimenty byl použit systém s procesorem AMD Opteron 2220 taktovaný na 2,8 GHz. V tabulce 8.4 lze vidět dobu potřebnou pro procházení celého stavového prostoru úlohy na tomto systému pro různý počet testovacích vektorů a scan registrů.

První tři řádky tabulky 8.4 byly odměřeny, hodnoty pro další řádky tabulky byly stanoveny extrapolací s využitím vztahu z tabulky 8.3. Poslední řádek tabulky 8.4 odpovídá obvodu b15 (podmnožina procesoru 80386), což byl největší obvod, jež byl pro experimenty k dispozici. Při použití plného scanu bylo pro tento obvod vygenerováno 1297 testovacích vektorů a počet scan registrů dosáhl 416. Z tabulky 8.4 je zřejmé, že úplné prohledání stavového prostoru u tohoto obvodu nepřipadá v úvahu. Pomocí GA bylo u tohoto obvodu dosaženo redukce příkonu na 93 % (viz tabulka 8.11). Výsledky jiných metod nebyly pro tento obvod k dispozici.

Tabulka 8.4: Časová náročnost procházení celého stavového prostoru úlohy pro různé obvody.

Počet testovacích vektorů	Počet scan registrů	Doba výpočtu	
8	3	17,9	minut
9	3	2,7	hodin
10	3	26,8	hodin
12	3	147,4	dní
15	3	1102,5	roků
1297	416	$4,71.10^{4374}$	roků

Tabulka 8.5: Výsledky dosažené na syntetické úloze.

Režim	Vel. pop.	Ukon. konv.	Konv. gen.	Počet gen.	Nejlepší řešení		Průměr 100 běhů		
					r [%]	t [s]	r_{avg} [%]	t_{all} [s]	n_{best} [%]
BF	–	–	–	–	55,2	1071,252	55,2	107125	100
AG	100	–	–	120	55,2	0,067	58,3	6	61
MG	800	–	–	2000	55,2	8,276	55,2	808	100
AK	100	0,99	50	85	55,2	0,047	59,0	5	18
MK	100	0,99	2016	2006	55,2	1,215	57,7	106	76

Z důvodů velké časové náročnosti procházení celého stavového prostoru úlohy byla vytvořena jednoduchá syntetická úloha, která umožnila analyzovat kvalitu výsledků dosažených pomocí GA. Úloha je volena tak, aby bylo možné v únosném čase prověřit všechny možnosti řešení a nalézt nejlepší možné řešení, které bude použito jako etalon pro ohodnocení kvality výsledků dosažených pomocí GA. Syntetickou úlohu tvoří jednoduchý obvod, jež obsahuje 3 registry scan, pro který bylo vygenerováno 8 testovacích vektorů.

Dosažené výsledky pro různé režimy prohledávání stavového prostoru (viz kapitola 7.2) lze vidět v tabulce 8.5. V tabulce 8.5 je uvedeno vždy nejlepší řešení získané během 100 běhů (r), průměrné řešení získané během 100 běhů (r_{avg}), doba běhu, během kterého bylo získáno nejlepší řešení (t), celkový čas pro 100 běhů (t_{all}) a procentní počet případů, ve kterých byl získán nejlepší výsledek (n_{best}). Dále je v tabulce uvedena velikost populace GA, jež byla nastavena automaticky nebo uživatelem. Pro režimy ukončující počtem generací je uveden počet generací. Pro režimy ukončující konvergencí je uvedena cílová konvergence a počet generací, přes které je konvergence počítána, dále je uveden počet generací, jež byly odsimulovány než bylo dosaženo ukončující podmínky.

V prvním řádku tabulky 8.5 jsou uvedeny výsledky pro procházení celého stavového prostoru úlohy. Nejlepší možné řešení tedy dosahuje redukce $r = 55,2\,\%$, toto řešení bylo nalezeno za 1071,252 s. Toto řešení bylo nalezeno v každém běhu (ve 100 % běhů). V tomto režimu nebylo použito GA, proto nejsou vyplněny sloupce týkající se parametrů GA. Výsledky dosažené v tomto režimu jsou použity jako etalon pro ostatní režimy.

Ve druhém řádku tabulky 8.5 jsou uvedeny výsledky pro automatický režim ukončující počtem generací. Pro tento režim bylo automaticky přednastaveno 120 generací a velikost populace 100 jedinců. V tomto režimu se taktéž podařilo nalézt nejlepší řešení, které bylo

nalezeno v 61 % běhů a pro nalezení tohoto řešení stačilo 67 ms. Průměr všech dosažených redukcí ze 100 běhů GA činí $r_{avg} = 58,3\,\%$.

Ve třetím řádku tabulky 8.5 jsou uvedeny výsledky pro manuální režim ukončující počtem generací. V tomto režimu byla nastavena velikost populace na 800 jedinců a počet generací na 2000, což jsou hodnoty, jež byly schválně zvoleny několikanásobně krát větší, než byly hodnoty přednastavené v automatickém režimu. Nalezení prvního řešení v tomto režimu trvalo 8, 2 s. Nejlepší řešení bylo nalezeno v každém běhu GA.

Ve čtvrtém řádku tabulky 8.5 jsou uvedeny výsledky pro automatický režim ukončující konvergencí. Ukončující konvergence byla přednastavena na 0,99 a byla počítána přes 50 posledních generací. Nejlepší řešení bylo nalezeno v 18 % případů, nalezení tohoto řešení trvalo 47 ms. Průměrný počet generací, po jejichž odsimulování bylo dosaženo ukončující podmínky, byl 85.

V posledním řádku tabulky 8.5 jsou uvedeny výsledky pro manuální režim ukončující konvergencí. Bylo přednastaveno počítání konvergence přes posledních 2000 generací, čímž bylo zajištěno, že bude odsimulováno nejméně 2000 generací (detaily viz kapitola 7.2). V tomto případě bylo nejlepší řešení nalezeno v 76 % případů, nalezení prvního tohoto řešení trvalo 1,215 s. Průměrný počet generací, po jejichž odsimulování bylo dosaženo ukončující podmínky, byl 2006.

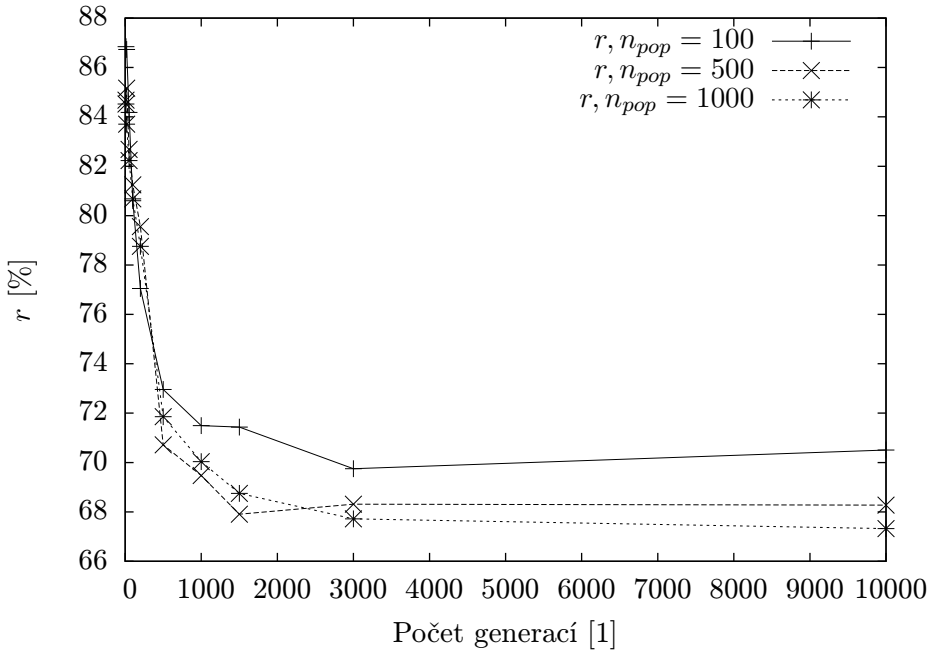
Z výše uvedeného experimentu je patrné, že GA poměrně rychle konverguje, takže stačí i malý počet generací pro nalezení použitelných řešení, jež nejsou výrazně horší než nejlepší existující řešení. Režim AG umožňuje bez jakéhokoliv zásahu uživatele nacházet použitelná řešení v rozumném čase. V uvedeném příkladu dokonce v 61 případech ze 100 dokonce i řešení, jež je nejlepší ze všech možných řešení. V režimu MG lze úspěšnost GA vylepšit. Platí, že čím větší populaci a větší počet generací je možné zvolit, tím více se vylepší úspěšnost GA. Při volbě počtu generací a velikosti populace jsme však zpravidla omezeni dostupným výpočetním výkonem. Režimy ukončující konvergencí jsou rychlé, nicméně dosažené výsledky již nejsou tak dobré, pravděpodobně z důvodu konvergence k lokálním extrémům, k jejichž opuštění nestihne dojít z důvodu vyhodnocení dosažené konvergence jako splnění ukončující podmínky.

8.3 Vliv parametrů GA

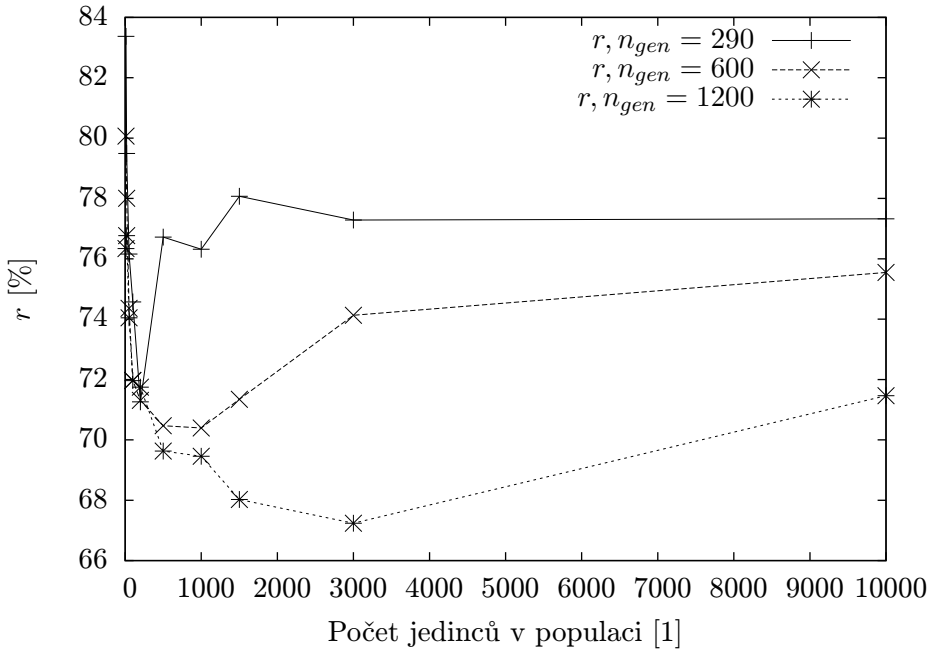
Cílem těchto experimentů bylo ověřit vliv parametrů GA na kvalitu nalezeného řešení a rychlost konvergence GA. Byl ověřován vliv velikosti populace a počtu generací. Pro následující experimenty byl použit obvod b02.

V grafu na obrázku 8.1 je zobrazen průběh průměrné redukce z deseti běhů GA pro různý počet generací. Na ose x je vynesen počet použitých generací GA, na ose y je vynesena průměrná dosažená redukce. V grafu je zobrazeno několik průběhů, každý pro různou velikost populace. Lze vidět, že GA poměrně rychle konverguje a kvalitní výsledky lze získat i pro malou populaci a několik set generací.

V grafu na obrázku 8.2 je zobrazen průběh průměrné redukce z deseti běhů GA pro různou velikost populace při konstantním počtu generací. Na ose x je vynesena velikost použité populace GA, na ose y je vynesena průměrná dosažená redukce. V grafu je zobrazeno několik průběhů, každý pro různý počet generací GA. Z grafu je patrné, že velikost redukce vzrůstá s rostoucím počtem jedinců v populaci až do dosažení jisté kritické hodnoty, od které redukce začne klesat, až se nakonec ustálí na určité hodnotě. Průběh je závislý na počtu použitých generací. U zkoumaného obvodu byla tato optimální velikost populace pro 290 generací v rozsahu 100 až 300 a pro 600 generací se pohybovala v rozsahu 500 až 1000.



Obrázek 8.1: Závislost hodnoty r na počtu generací GA pro obvod b02.



Obrázek 8.2: Závislost hodnoty r na velikosti populace GA pro obvod b02.

Tabulka 8.6: Dosažené r při použití různých metrik pro optimalizaci a simulaci, obvod s349.

		Metrika pro simulaci			
		Hamming	**NTC**	**WNTC**	**WSA**
Metrika pro optimalizaci	**Hamming**	**79,6**	92,6	93,3	93,6
	NTC	90,3	**70,1**	71,7	71,1
	WNTC	95,8	72,6	**73,8**	73,0
	WSA	94,1	73,0	74,5	**73,9**

8.4 Vliv použité metriky

Cílem následujícího experimentu bylo ověřit vztahy mezi jednotlivými metrikami. V tabulce 8.6 jsou uvedeny výsledky pro obvod s349, ale obdobných výsledků bylo dosaženo i pro ostatní obvody. Tabulka 8.6 zobrazuje dosaženou redukci pro různé metriky, přičemž vždy bylo provedeno 20 běhů GA. V řádcích tabulky jsou uvedeny metriky, jež byly použity pro výpočet fitness jedinců populace během optimalizace pomocí GA. Výstupem GA byl návrh pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce scan. Toto navržené pořadí bylo dále odsimulováno při použití metrik, jež jsou uvedeny ve sloupcích. Tučně zvýrazněná diagonála tabulky tedy představuje stav, kdy metrika použitá pro optimalizaci i pozdější simulaci, byla shodná. V tomto případě lze vidět, že výsledky při použití jednotlivých metrik si přibližně odpovídají, přičemž nejnižší redukce bylo dosaženo při použití výpočtu Hammingovy vzdálenosti mezi testovacími vektory (79,6 %) a nejvyšší redukce při použití metriky NTC (70, 1 %). Z tabulky je dále zřejmé, že rozdíl mezi metrikami NTC, WNTC a WSA je téměř zanedbatelný. Situace se však změní, pokud pořadí získané s využitím výpočtu Hammingovy vzdálenosti mezi testovacími vektory odsimulujeme s metrikou NTC. V tomto případě bude dosažená redukce pouze 92,6 % původní hodnoty a obdobně i pro metriky WNTC a WSA. Podobná situace nastává, pokud pro pořadí získané s využitím metrik NTC, WNTC nebo WSA, vypočteme hodnotu redukce pomocí Hammingovy vzdálenosti mezi testovacími vektory. Metriky NTC, WNTC a WSA se mezi sebou jeví jako zaměnitelné. **Výše uvedené demonstruje, že minimalizací Hammingovy vzdálenosti mezi testovacími vektory nemusí být minimalizován příkon obvodu během aplikace testu.** Vzhledem k tomu, že toto bylo demonstrováno na obvodu ze standardní benchmarkové sady, je zřejmé, že obvodů, u kterých nebudou optimalizace využívající minimalizace Hammingovy vzdálenosti mezi testovacími vektory nacházet řešení dostatečné kvality, bude jistě existovat velké množství. Pro korektní prokázání uvedeného tvrzení by však bylo třeba provést měření na reálných obvodech, což by však bylo velice nákladné. Toto měření by jistě potvrdilo vyšší korelaci metrik NTC, WNTC a WSA s reálným průběhem příkonu než v případě využití výpočtu Hammingovy vzdálenosti mezi testovacími vektory.

8.5 Porovnání sekvenční a souběžné optimalizace

Cílem tohoto experimentu bylo porovnat výsledky dosažené sekvenční a souběžnou optimalizací. V případě sekvenční optimalizace je nejdříve optimalizováno pořadí aplikace testovacích vektorů a následně je optimalizováno pořadí zapojení registrů do řetězce scan (nebo naopak). Implicitně se předpokládá souběžná optimalizace. V tomto případě se současně optimalizuje pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce

Tabulka 8.7: Porovnání sekvenční a souběžné optimalizace pro vybrané obvody z ISCAS89.

Obvod	r_{vec} [%]	r_{seq} [%]	t_{seq} [s]	r_{par} [%]	t_{par} [s]
s27	77,7	77,7	4,009	66,8	2,245
s298	82,1	82,1	1783,265	75,0	863,891
s344	89,0	86,7	1462,634	83,7	713,631
s349	78,1	75,7	1261,496	70,1	614,781
s382	85,1	82,4	4729,337	73,3	2292,118
s386	81,7	78,4	4748,800	70,8	2297,434
s444	80,2	76,4	4505,608	64,6	2122,130

scan a v rámci řešení optimalizační úlohy se prochází jen jeden stavový prostor. Pro tento experiment byl použit výpočetní systém se dvěma procesory AMD Opteron 2220 dual core taktovaný na 2,8 GHz. Porovnání obou přístupů lze vidět v tabulce 8.7, která obsahuje název obvodu, hodnotu redukce při optimalizaci pořadí aplikace testovacích vektorů (r_{vec}), hodnotu redukce při provedení optimalizace pořadí zapojení registrů scan po optimalizaci pořadí aplikace testovacích vektorů, neboli sekvenční optimalizace (r_{seq}), délku trvání sekvenční optimalizace (t_{seq}), hodnotu redukce při provedení obou optimalizací souběžně (r_{par}) a délka trvání souběžné optimalizace (t_{par}). Pro každý případ bylo provedeno 10 běhů GA, ze kterých byly vybrány nejlepší dosažené výsledky. Ve všech případech bylo při použití souběžné optimalizace dosaženo lepších výsledků (nižší hodnota r). **Tímto experimentem bylo potvrzeno, že souběžnou optimalizací lze dosáhnout lepších výsledků, než při aplikaci obou optimalizací postupně po sobě, což byl jeden z předpokladů při návrhu metodiky. Další výhodou je, že souběžná optimalizace vyžaduje jen polovinu času při srovnání se sekvenční optimalizací**. V případě použití sekvenční optimalizace v opačném pořadí (tedy optimalizace pořadí aplikace testovacích vektorů po optimalizaci pořadí zapojení registrů do řetězce scan) jsou výsledky obdobné a nejsou proto uváděny.

8.6 Porovnání s existujícími metodami

Tato kapitola porovnává dosažené výsledky s výsledky existujících metod. Objektivní porovnání je však problematické vzhledem k nejrůznějším odlišnostem u jednotlivých metod. Publikované metody totiž využívají mapování na různé technologie, různé generátory testovacích vektorů s různým nastavením i různý způsob vyhodnocení dosažených výsledků. Tyto informace často nejsou k dispozici, takže nelze zajistit stejné podmínky pro provedení experimentů.

V tabulce 8.8 lze vidět porovnání navržené metody s metodou publikovanou v [30] pro vybrané obvody z benchmarkových sad ISCAS85 a ISCAS89. Do porovnání byly zahrnuty vždy nejlepší dosažené výsledky z přibližně dvaceti různých běhů GA při kombinaci režimů MG/AG. Metoda publikovaná v [30] pracuje s Hammingovou vzdáleností mezi jednotlivými testovacími vektory. Tato metoda umožňuje sekvenční optimalizaci pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězců scan. Metoda konvertuje problém na problém obchodního cestujícího, který řeší pomocí GA. Pro tuto metodu nebyly k dispozici žádné bližší informace o nástrojích a nastaveních použitých pro získání publikovaných výsledků. Takže nelze ani odvodit informace o počtu vygenerovaných testovacích

Tabulka 8.8: Porovnání navržené metody s metodou publikovanou v [30] pro vybrané obvody z ISCAS85 a ISCAS89.

	Jelodar, HV TSP GA	**Mentor, AMI** $0.5\,\mu m$				
Obvod	r_1 [%]	**Počet** TC	FC [%]	r_2 [%]	r_{2HV} [%]	r_{2SHV} [%]
c432	69,0	81	98,64	**64,8**	**59,9**	84,9
c880	80,0	95	100,00	**75,0**	**75,1**	90,9
c1355	84,4	69	99,80	**80,0**	**68,0**	89,6
c1908	65,7	107	99,46	75,1	**63,7**	81,5
c2670	90,1	59	59,22	**86,4**	**88,6**	94,0
c7552	91,5	332	99,87	**91,2**	**91,1**	98,9
s298	58,2	50	96,87	75,0	79,7	96,7
s444	68,9	65	97,07	**64,6**	73,5	92,7
s641	77,0	88	99,13	**65,9**	79,9	91,0
s1423	84,7	133	99,52	**74,8**	**82,3**	95,5
s1488	58,7	156	88,50	70,6	72,5	88,6
s5378	90,3	309	99,16	**85,3**	**89,0**	99,1

vektorů a o dosaženém pokrytí chyb, což jsou důležité parametry výrazně ovlivňující dosažené výsledky. Publikované hodnoty redukce příkonu byly přepočítány pomocí vztahu $r = 100 - publikovana\_hodnota$, protože daná publikace vyhodnocuje výsledky jako „redukce o X procent původní hodnoty“ místo v této práci uvažovaného vyhodnocení jako „redukce na X procent původní hodnoty“ (označení symbolem r).

Tabulka 8.8 obsahuje název obvodu, hodnoty redukce po přepočítání z „úspory“ na redukci (označení r_1) pro metodu publikovanou v [30]. V dalších sloupcích tabulky jsou uvedeny výsledky dosažené s navrženou metodou. Jmenovitě ve třetím sloupci tabulky 8.8 jsou uvedeny počty použitých testových cyklů. V každém testovém cyklu je aplikován jeden testovací vektor a v případě obvodů s plným scanem i jeden scan vektor. Další sloupec tabulky obsahuje pokrytí chyb pro použitou sadu testovacích vektorů. Dále jsou uvedeny dosažené hodnoty redukce (označení r_2) při použití příkonové metriky NTC. Aby bylo možné lépe srovnat úspěšnost metod při procházení stavového prostoru úlohy, jsou pro navrženou metodu uvedeny i dosažené hodnoty redukce při použití Hammingovy vzdálenosti mezi testovacími vektory (označení r_{2HV}). V posledním sloupci tabulky jsou uvedeny hodnoty redukce (označení r_{2SHV}) po simulaci aplikace testu s metrikou NTC pro pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce scan odpovídající výsledkům prezentovaným ve sloupci označeném r_{2HV}. Tučně jsou zvýrazněny případy, ve kterých navržená metoda dosáhla vyšší redukce.

U obvodu c2670 bylo při generování testu dosaženo pokrytí chyb pouze $59,22\,\%$. Zvýšení pokrytí chyb u tohoto obvodu by bylo možné dosáhnout vložením testovacích bodů (viz kapitola 2.6). U všech ostatních obvodů bylo dosaženo pokrytí chyb nad $80\,\%$. Pokud srovnáme sloupce označené r_1 a r_2, tak ve většině případů bylo dosaženo vyšší hodnoty redukce než u metody publikované v [30]. Pokud srovnáme sloupce označené r_1 a r_{2HV}, tak ve většině případů bylo také u prezentované metody dosaženo vyšší hodnoty redukce, přestože pracovala pouze s Hammingovou vzdáleností mezi testovacími vektor. Např. v případě obvodu c7552 jsou hodnoty r_1 a r_{2HV} téměř shodné ($r_1 = 91,5\,\%$, $r_{2HV} = 91,1\,\%$), nicméně pokud takto

nalezené pořadí aplikace testovacích vektorů využijeme při simulaci s využitím metriky NTC, dosáhneme redukce pouze $r_{2SHV} = 98,9\,\%$. Proto lze předpokládat, že kvalitativně budou výsledky prezentované ve sloupci r_2 mnohem lepší než výsledky prezentované ve sloupci r_1 (viz kapitola 8.4) a je tedy možné, že prezentovaná metoda byla úspěšnější ještě ve větším množství případů.

Pro další porovnání byly použity metody nepoužívající Hammingovu vzdálenost mezi testovacími vektory, k nimž bylo publikováno více informací. Tabulka 8.9 porovnává dosažené výsledky s výsledky publikovanými ve [22]. Do porovnání byly zahrnuty vždy nejlepší dosažené výsledky z přibližně dvaceti různých běhů GA. Tabulka obsahuje název obvodu a počet testovacích vektorů použitých pro každou metodu (u kombinačních obvodů je shodný s počtem TC). Původní NTC obvodu stanovené metodou publikovanou ve [22] je v tabulce označeno jako NTC_1. NTC po redukci je označeno jako NTC_{opt1} a dosažená redukce je označena jako r_1. Hodnota r_1 byla vypočtena z publikovaných údajů s využitím rovnice (4.1) jako $\frac{NTC_{opt1}}{NTC_1}100$, protože ve [22] tento údaj není přímo uveden. FC značí procentní pokrytí chyb použité sady testovacích vektorů. Pro metodu z [22] nebyl tento údaj k dispozici, proto je nemožné srovnat kvalitu dosažených výsledků z hlediska pokrytí chyb. NTC_{opt2} značí NTC po redukci prezentovanou metodou. Hodnoty r_2 představují redukce dosažené prezentovanou metodou. NTC před redukcí není uváděno, protože jej lze snadno odvodit z rovnice (4.1).

Hodnoty NTC jsou u obou metod odlišné z důvodu použití různých technologických knihoven. Koeficient r je však již porovnatelný. Navržená metoda dosáhla ve většině případů lepších výsledků (nižší r) a v jednom případě dosáhla horšího výsledku. Pro většinu obvodů byla metoda spuštěna v režimu AG, pouze u obvodu c6288 byl použit režim MG a ručně zdvojnásoben počet generací ng_{set}. Výhoda současné optimalizace pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce scan se mohla projevit pouze u obvodu b12, protože ostatní obvody jsou kombinační. Nejlepšího výsledku při srovnání s publikovanou metodou bylo dosaženo u obvodu b12 (nejnižší r_2/r_1). Nejhoršího u obvodu c7552 (nejvyšší r_2/r_1). Tento výsledek se nepodařilo vylepšit ani při použití režimu MG. Důvod tak značně odlišnosti výsledků u obvodu c7552 se nepodařilo zjistit. Nejvyšší redukce bylo dosaženo u obvodu b12 ($r_2 = 44,4\,\%$), nejnižší u obvodu c7552 ($r_2 = 91,2\,\%$). Naproti tomu metoda publikovaná v [22] dosáhla nejvyšší redukce u obvodu c1908 ($r_1 = 61,4\,\%$) a nejnižší redukce u obvodu c6288 ($r_1 = 91,8\,\%$). U prezentované metody by bylo pravděpodobně možné dosáhnout i lepších výsledků při ručním vyladění parametrů GA pro daný případ. Detailnější porovnání je problematické, proto nejsou známy konkretní testovací vektory ani dosažené pokrytí chyb, se kterým pracuje porovnávaná metoda. Vzhledem k nižšímu počtu testovacích vektorů však lze předpokládat, že pokrytí chyb bude zřejmě nižší. Je tedy pravděpodobné, že s takovými vstupními daty by implementovaná metoda dosáhla ještě vyšší redukce.

Tabulka 8.10 porovnává dosažené výsledky s výsledky publikovanými v [11]. Význam sloupců tabulky je obdobný jako u tabulky 8.9. Z výsledků publikovaných v 8.10 byly pro porovnání zvoleny výsledky pro neizolovatelné (viz kapitola 4.2.2) registry scan a nulový model zpoždění, které nejblíže odpovídají prezentované metody. V případě porovnání s ostatními variantami je dosaženo obdobných výsledků. Pokrytí chyb ani počet použitých testovacích vektorů nebyl u porovnávané metody k dispozici. U všech obvodů kromě obvodu s27 bylo s navrženou metodou dosaženo lepších výsledků (nižší r). Výrazný rozdíl u obvodu s27 se opět nepodařilo objasnit. Ve všech případech byla implementovaná metoda spuštěna v režimu AG. Protože se jedná o sekvenční obvody, tak se mohla projevit výhoda souběžné optimalizace pořadí aplikace testovacích vektorů a zapojení registrů do řetězce scan, díky

Tabulka 8.9: Porovnání navržené metody s metodou publikovanou v [22] pro vybrané obvody z ISCAS85, ITC99.

	Girard, HITEST, ES2, heuristika				**AG/MG, Mentor, AMI** $0.5\,\mu m$			
Obvod	**Počet vektorů**	NTC_1	NTC_{opt1}	r_1 [%]	**Počet** TC	FC [%]	NTC_{opt2}	r_2 [%]
c880	27	7140	6403	89,7	95	100,00	7487	**75,0**
c1355	89	33181	28511	85,9	69	99,80	5945	**80,0**
b12	33	7214	4429	61,4	299	98,83	2932339	**44,4**
c1908	69	48826	38250	78,3	107	99,46	7872	**75,1**
c3540	82	82241	67603	82,2	233	98,86	67204	**81,3**
c5315	56	99154	90101	90,9	173	99,61	102950	**90,1**
c6288	41	462018	424086	91,8	69	99,96	25681	**91,0**
c7552	72	232123	159414	68,7	332	99,87	237008	91,2

Tabulka 8.10: Porovnání navržené metody s metodou publikovanou v [11] pro vybrané obvody z ISCAS89.

	Chakravarty, ATALANTA greedy search			**AG, Mentor, AMI** $0.5\,\mu m$			
Obvod	NTC_1	NTC_{opt1}	r_1 [%]	**Počet** TC	FC [%]	NTC_{opt2}	r_2 [%]
s27	49	22	44,9	11	83,06	166	66,8
s298	28644	26034	90,9	50	96,87	33186	**75,0**
s344	20440	18923	92,6	46	97,44	19610	**83,7**
s349	20790	19088	91,8	133	83,59	14293	**70,1**
s382	58667	54147	92,3	773	84,69	47437	**73,3**
s386	7996	6715	84,0	100	98,09	38413	**70,8**
s444	66186	63050	95,3	65	97,07	41851	**64,6**

čemuž bylo pravděpodobně ve většině případů dosaženo vyšší redukce. Nejlepšího výsledku při srovnání s metodou publikovanou v [11] bylo dosaženo u obvodu s444 (nejnižší r_2/r_1). Nejhoršího u obvodu s27 (nejvyšší r_2/r_1). Nejvyšší redukce bylo dosaženo u obvodu s444 ($r_2 = 64,6\,\%$), nejnižší u obvodu s344, $r_2 = 83,7\,\%$. Naproti tomu metoda z [11] dosáhla nejvyšší redukce u obvodu s27 ($r_1 = 44,9\,\%$) a nejnižší redukce u obvodu s444 ($r_1 = 95,3\,\%$).

8.7 Další dosažené výsledky

Tabulka 8.11 obsahuje výsledky dosažené na podmnožině obvodů z benchmarkové sady ITC99. Tyto výsledky jsou uvedeny bez srovnání, protože se nepodařilo nalézt žádné publikované výsledky, jež by bylo možné použít pro srovnání. Jedním z možných vysvětlení je vyšší složitost této benchmarkové sady ve srovnání s předchozími a rozsáhlý prostor řešení. V tabulce 8.11 je uveden počet prvků obvodu po namapování na technologickou knihovnu a vložení plného scanu, počet vstupů a výstupů obvodu (PI/PO) po vložení plného scanu (včetně všech řídicích a synchronizačních vstupů), počet registrů (FF), nárůst plochy čipu po vložení plného scanu. Význam ostatních symbolů je obdobný jako v tabulce 8.10. Protože

Tabulka 8.11: Výsledky dosažené s obvody z benchmarkové sady ITC99.

Obvod	**Počet prvků**	**Počet PI/PO**	**Počet FF**	**Nárůst plochy**	**Počet** TC	FC [%]	NTC_{opt}	r [%]
b01	39	9/2	5	1,14	24	77,09	1713	68,3
b02	24	8/1	4	1,18	25	70,99	757	67,2
b03	122	11/4	30	1,14	52	96,83	59044	75,0
b04	588	18/8	66	1,09	101	95,13	747610	59,6
b05	618	8/37	34	1,05	181	91,50	867756	73,4
b06	58	9/6	8	1,13	32	88,81	4267	66,7
b07	350	8/8	41	1,09	117	96,30	445179	74,7
b08	130	16/4	21	1,12	81	97,29	43248	62,5
b09	215	8/1	28	1,10	68	98,21	136552	75,3
b10	189	18/6	17	1,08	97	95,51	100269	74,6
b11	546	14/6	31	1,05	160	92,65	1039014	88,1
b12	955	12/6	119	1,09	299	98,83	2932339	44,4
b13	281	17/10	49	1,11	100	97,52	444572	81,0
b14	5109	39/54	215	1,04	620	96,61	98169725	87,6
b15	8675	43/70	417	1,04	1297	96,44	249128549	93,0

nejsou žádná data pro srovnání, koeficient r je uváděn bez indexu. Pro většinu obvodů byla metoda spuštěna v režimu AG, jen pro obvody b14, b15 byl vzhledem ke značné složitosti těchto obvodů použit režim MG se sníženou velikostí populace a počtu generací, aby bylo výsledku dosaženo rychleji. Z tohoto důvodu by pravděpodobně mohlo být dosaženo i vyšší redukce.

V následujícím experimentu byl zkoumána metriky WSA. Pro tento experiment byl použit výpočetní systém se dvěma procesory AMD Opteron 2220 dual core taktovaný na 2,8 GHz. Dosažené výsledky jsou uvedeny v tabulce 8.12.

V tabulce 8.12 jsou uvedeny výsledky pro několik benchmarkových obvodů při použití metriky WSA. V tabulce je uvedena hodnota WSA před optimalizací a po optimalizaci (WSA_{opt}), dosažená redukce (r) a čas výpočtu (t). Protože metrika WSA uvažuje rozvětvení a kapacitu, je v tabulce 8.12 uvedena i průměrná kapacita vstupů daného obvodu (Cin_{avg} ve femto faradech) a průměrné rozvětvení výstupů (FO_{avg}). Metrika WSA je nejpřesnější z metrik, kterou implementace podporuje. Nicméně pokud výsledky srovnáme s tabulkami 8.10 a 8.11, zjistíme, že dosažené výsledky jsou porovnatelné s výsledky získanými při použití metriky NTC, která je však výpočetně méně náročná.

8.8 Škálovatelnost úlohy

U zkoumaných obvodů bylo možné získat kvalitní výsledky (při porovnání s ostatními metodami) v únosném čase při použití moderního výpočetního systému. Vzhledem k předpokladu, že se tato optimalizace bude provádět jen jednou během životního cyklu obvodu a to až v poslední fázi jeho vývoje, není ve většině případů vyšší časová náročnost výraznou nevýhodou, obzvlášť při dosažení kvalitnějších výsledků. Navíc implementace podporuje multiprocesorové systémy, takže v případě dostupnosti takového systému lze výpočet značně urychlit.

Tabulka 8.12: Výsledky dosažené pro vybrané benchmarkové obvody při použití metriky WSA.

Obvod	Cin_{avg} [**fF**]	FO_{avg}	WSA	WSA_{opt}	r [%]	t [s]
s27	14,691	1,148	214,5	145,6	67,9	2,321
s298	16,648	1,852	47413,4	37852,8	79,8	867,010
s344	15,937	1,803	28241,2	24480,2	86,7	714,826
s349	15,996	1,809	22414,8	16564,5	73,9	616,456
s382	16,334	1,903	67192,4	51791,3	77,1	2322,522
s386	16,565	1,864	55742,6	39240,1	70,4	2309,653
s444	16,532	1,923	71537,9	47215,0	66,0	2279,354
b01	11,539	1,868	2913,0	1900,7	65,3	29,251
b02	17,717	1,806	1182,0	832,5	70,4	18,054
b03	17,493	2,227	96885,3	78356,4	80,9	2409,439

Tabulka 8.13: Škálovatelnost úlohy, obvod b03.

Počet CPU	**Doba výpočtu** [s]	**Zrychlení**	**Režie paralelního výpočtu** [%]
1	8258,839	1,000	–
2	5679,108	1,454	27,29
3	4055,467	2,036	32,12
4	3194,716	2,585	35,37
5	2562,390	3,223	35,54
6	2180,236	3,788	36,87
7	1908,748	4,327	38,19
8	1716,419	4,812	39,85

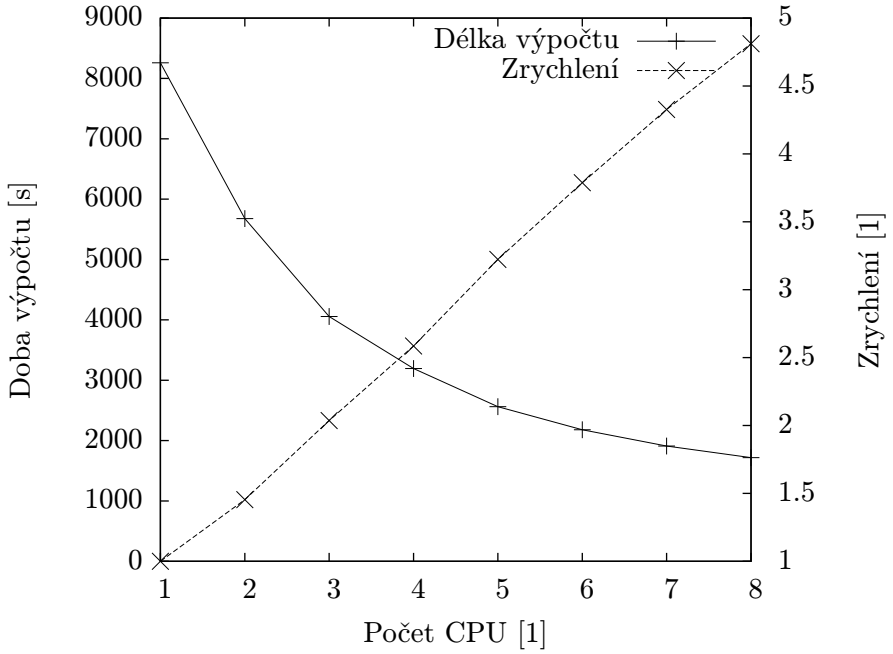
Pro následující experiment byl použit výpočetní systém se dvěma čtyř jádrovými procesory Intel Xeon X5355 s taktem 2,66 GHz. Cílem experimentu bylo ověřit škálovatelnost úlohy na reálném multiprocesorovém systému. V tabulce 8.13 jsou uvedeny výsledky dosažené pro obvod b03.

V prvním sloupci tabulky 8.13 je uveden počet CPU použitých pro výpočet. Ve druhém sloupci jsou uvedeny doby trvání výpočtů v sekundách a ve třetím sloupci dosažené zrychlení, které je vypočteno jako $doba\_vypoctu\_1/doba\_vypoctu\_n$, kde $doba\_vypoctu\_1$ je doba výpočtu při použití jednoho CPU a $doba\_vypoctu\_n$ je doba výpočtu při použití n CPU. Čtvrtý sloupec uvádí režii paralelního výpočtu pro daný počet CPU, která je vypočtena jako $(1 - zrychleni/pocet\_cpu)100$. V případě použití jednoho CPU nemá smysl uvažovat režii paralelního výpočtu.

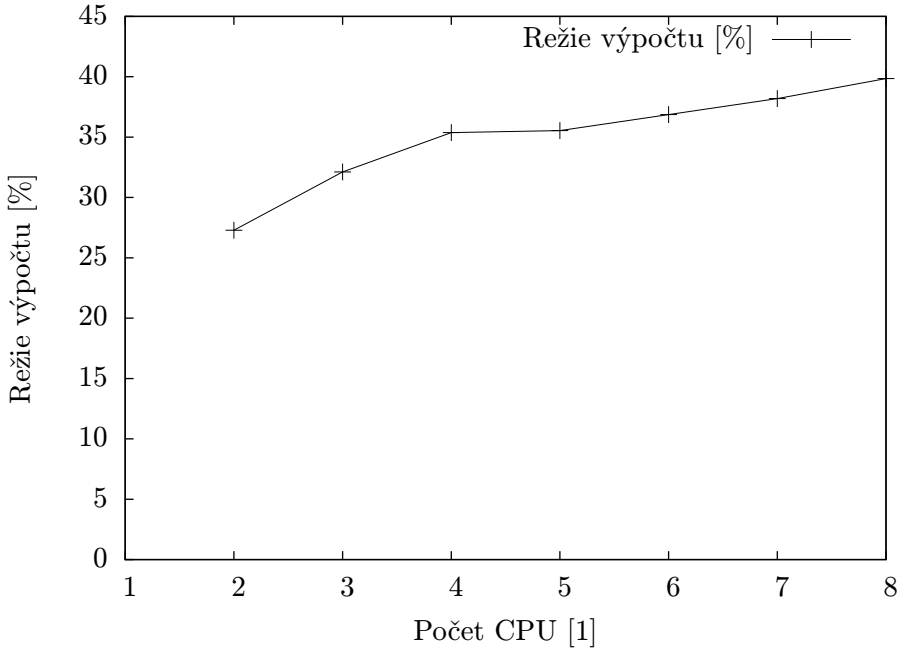
Doba výpočtu a zrychlení z tabulky 8.13 jsou vyneseny do grafu 8.3. Na ose x je vynesen počet CPU, na hlavní ose y je vynesena doba výpočtu v sekundách a na vedlejší ose y zrychlení. V grafu na obrázku 8.3 lze vidět, že zrychlení vzrůstá téměř lineárně, pro 8 CPU je dosažené zrychlení rovno $4,812$.

Režie paralelního výpočtu v závislosti na počtu CPU je vynesena do grafu 8.4. Na ose x je vynesen počet CPU, na ose y je vynesena režie paralelního výpočtu.

V grafu na obrázku 8.4 lze vidět, že režie paralelního výpočtu v případě použití osmi CPU roste téměř lineárně. Pro 8 CPU je režie paralelního výpočtu rovna $39,85\,\%$. Do této



Obrázek 8.3: Zrychlení při použití multiprocesorového systému, obvod b03.



Obrázek 8.4: Režie paralelního výpočtu při použití multiprocesorového systému, obvod b03.

režie je ovšem započítáno i načítání knihoven, verifikace obvodu, generování vyhledávacích tabulek pro simulaci a prvotní simulace pro stanovení PWR_{orig} (viz rovnice (7.1)), takže vlastní komunikační režie bude ještě nižší.

Kapitola 9

Závěr

Tato práce se zabývá redukcí příkonu obvodu během aplikace testu. Nejdříve byly analyzovány příčiny zvýšeného příkonu obvodu během aplikace testu při srovnání s běžným funkčním režimem činnosti. Při této analýze byla uvažována technologie CMOS jakožto dominantní technologie pro výrobu moderních číslicových obvodů. Byly definovány zjednodušující metriky pro ohodnocení příkonu obvodu během aplikace testu. Dále byla provedena analýza aktuálního stavu řešení této problematiky. Existující metody byly rozčleněny dle stanovených kritérií. Pozornost byla zaměřena zejména na metody zabývající se optimalizací pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězců scan pro snížení dynamické příkonové složky během aplikace testu. Výhodou těchto metod je snadná možnost začlenění do existujícího návrhového procesu. Tyto metody jsou též použitelné i pro obvody vyráběné nejnovějšími realizačními technologiemi, u kterých převládá statická příkonová složka nad dynamickou. U takových obvodů je ovšem účinnost těchto metod snížena.

Na základě výsledků provedené analýzy byla identifikována slabá místa existujících metod pro optimalizaci pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězce scan. Hlavním nedostatkem většiny těchto metod je odstup od realizační technologie, kdy jsou při optimalizaci uvažována pouze vstupní data testu bez návaznosti na vnitřní strukturu obvodu. To je většinou proto, že nejsou k dispozici dostatečně přesné nástroje pro vyhodnocení příkonu obvodu během aplikace testu, které by byly schopny toto vyhodnocení provést v dostatečně krátkém času. Úprava stávajících simulačních nástrojů je problematická, protože tyto simulátory sice disponují velkou přesností, která je pro účely uvažované optimalizační úlohy nadbytečná, ale jsou velice pomalé. Tyto simulátory jsou totiž konstruovány pro provádění jednotlivých simulací většinou pro účely verifikace návrhu. Stavový prostor uvažované optimalizační úlohy je však obrovský, takže i při použití nejrůznějších heuristik je třeba simulací ohodnotit kvalitu velkého množství řešení, přičemž otázka přesnosti simulace je až druhořadá. Při využití současné výpočetní techniky by tak nasazení standardních simulačních nástrojů pro většinu existujících obvodů znamenalo nemožnost vyřešení úlohy v rozumném čase. Z tohoto důvodu většina existujících metod volí Hammingovu vzdálenost mezi testovacími vektory jakožto metriku pro ohodnocení kvality jednotlivých řešení. V této práci bylo ovšem předvedeno, že optimalizace, které si kladou za cíl minimalizaci Hammingovy vzdálenosti mezi testovacími vektory nevedou k minimalizaci příkonu obvodu během aplikace testu. Dalším nedostatkem existujících optimalizačních metod je rozdělení problematiky na dvě optimalizační úlohy, přičemž první úloha optimalizuje pořadí aplikace testovacích vektorů a následně druhá úloha optimalizuje pořadí zapojení registrů do řetězce scan. Při tomto způsobu řešení kvalita výsledků druhé optimalizační

úlohy závisí na výsledku první optimalizační úlohy, tedy mezi těmito optimalizačními úlohami existuje závislost. Jednotlivé optimalizační úlohy však spolu nijak nekomunikují, díky čemuž může dojít u první optimalizační úlohy k opomenutí některých řešení, které by se vzápětí u druhé optimalizační úlohy mohly ukázat jako výhodné.

Výše uvedené nedostatky se snaží řešit metodika, jež byla prezentována v této práci. Pro definici metodiky byl prezentován formální model testovaného číslicového obvodu. Nad tímto modelem byly potom definovány operátory a algoritmy, s jejichž pomocí byla vystavěna výsledná metodika. Metodika umožňuje současnou optimalizaci pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězců scan. Problém je díky zvolenému kódování řešen jako jedna optimalizační úloha s jedním stavovým prostorem. Metodika využívá vlastní simulátor aplikace testu, který je navázán na konkrétní realizační technologii. Simulátor je navrhnut tak, aby pracoval se zjednodušujícími příkonovými metrikami nad primitivy ze zvolené technologické knihovny. Díky tomu simulace probíhá mnohem rychleji než při použití standardních simulačních metod na fyzické úrovni návrhu a dosažené výsledky mnohem více odpovídají realitě než při použití Hammingovy vzdálenosti mezi testovacími vektory. Pro procházení stavového prostoru úlohy byl použit genetický algoritmus, pro který bylo definováno vhodné kódování problému.

V rámci řešení práce byla tato metodika úspěšně implementována. Zdrojové kódy implementace byly uvolněny pod licencí GNU GPL. Implementace je použitelná ve spojitosti s návrhovými nástroji firmy Mentor Graphics. Pro simulaci aplikace testu byly sestaveny technologické knihovny pro jednotlivé varianty technologie AMI. Implementace podporuje i v dnešní době stále více rozšířenější multiprocesorové systémy, u kterých umožňuje rozdělení výpočtu na jednotlivé procesory. Metodika byla ověřena nad obvody z benchmarkových sad ISCAS85, ISCAS89 a ITC99. Taktéž bylo provedeno srovnání s výsledky existujících metod. Byla demonstrována výhoda současné optimalizace pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězců scan nad sekvenční optimalizací pomocí dvou optimalizačních úloh. Pro uvedené obvody byl demonstrován vztah mezi jednotlivými příkonovými metrikami a Hammingovou vzdáleností mezi testovacími vektory. Bylo konstatováno, že výsledky získané s využitím Hammingovy vzdálenosti číselně odpovídají výsledkům získaným s využitím simulace, nicméně při opětovném odsimulování těchto výsledků pomocí některé příkonové metriky, vychází pak tyto výsledky mnohem hůře. Díky tomu lze usuzovat, že i reálný příkon obvodů během aplikace testu dle návrhu pořadí aplikace testovacích vektorů a zapojení registrů do řetězců scan provedeného metodou, která využívá Hammingovu vzdálenost mezi testovacími vektory, bude vyšší než při použití metody využívající simulace. Pro prokázání tohoto tvrzení by však bylo nutné provést měření na reálných obvodech, které je však ze svojí podstaty složité a nákladné. Bylo by nutné fyzicky realizovat jednotlivé obvody a odměřit průběhy příkonů pro jednotlivé aplikace testu. To by vyžadovalo velice rychlé a přesné měřící přístroje, protože přechody obvodu mezi jednotlivými stavy jsou velice rychlé, obzvlášť v případě testování na rychlosti čipu (at-speed testing).

Během řešení této práce byla metodika nasazena i ve spojitosti s tzv. Testovatelnými bloky (viz [78]), což umožnilo redukci stavového prostoru této optimalizační úlohy. Nevýhodou je, že rozdělení obvodu na Testovatelné bloky je možné jen u omezené množiny většinou syntetických číslicových obvodů, což výrazně limituje možnosti použití této kombinované metodiky.

Prostor pro vylepšení lze nalézt zejména u implementace. Implementovaná simulační metoda sice zohledňuje kapacitní zátěž jednotlivých obvodových prvků i strmost hran jednotlivých vstupních signálů, nicméně neuvažuje zpoždění jednotlivých obvodových prvků. Díky tomu nejsou uvažována překlopení v obvodu způsobená hazardy. Taktéž nejsou uva-

žovány zákmity, a proto lze implementaci aplikovat pouze na nekmitající obvody. Otázkou je, zda by při implementaci těchto vlastností převážila výhoda vyšší přesnosti nad vyšší časovou náročností simulace a zda by tato vyšší přesnost umožnila nacházení nějakých výhodnějších řešení optimalizační úlohy. Další možností vylepšení je kombinace s metodou využívající dodatečné testovací vektory (viz 4.3). Taktéž by bylo možné experimentovat s různými variantami genetického algoritmu, z nichž některé varianty by mohly být pro danou úlohu výhodnější. Implementaci by bylo možné dále rozšířit o migrující genetický algoritmus pro podporu multiprocesorových systémů obsahujících řádově stovky či tisíce procesorů.

9.1 Přínos práce

Přínos práce lze shrnout do následujících bodů:

1. Rozšíření existujícího formálního modelu číslicového obvodu pro popis testovaného číslicového obvodu.
2. Formální definice algoritmů pro simulátor aplikace testu, který kvantifikuje zvolenou příkonovou metriku během aplikace testu a jeho implementace.
3. Sestavení technologických knihoven pro implementovaný simulátor. Byly sestaveny knihovny pro technologie AMI $0,5\,\mu m$ a AMI $1,2\,\mu m$, každá ve variantě *slow*, *typ*, *fast*. s
4. Návrh metodiky využívající genetický algoritmus pro současnou optimalizaci pořadí aplikace testovacích vektorů a pořadí zapojení registrů do řetězců scan a její implementace.
5. Experimentální ověření navržené metodiky a porovnání s existujícími metodami pro benchmarkové sady ISCAS85, ISCAS89 a ITC99.

Seznam použité literatury

[1] ISCAS85 Benchmarks. [Online], [rev. 1. 12. 1993], [cit. 12. 1. 2008].
URL ftp://mcnc.mcnc.org/pub/benchmark/ISCAS85 96

[2] ISCAS89 Benchmarks. [Online], [rev. 1. 12. 1993], [cit. 12. 1. 2008].
URL ftp://mcnc.mcnc.org/pub/benchmark/ISCAS89 96

[3] ITC'99 Benchmarks Web Site. [Online], [rev. 7. 9. 2007], [cit. 12. 1. 2008].
URL http://www.cad.polito.it/tools/itc99.html 96, 97

[4] ABADIR, M. S.; BREUER, M. A.: A knowledge based system for designing testable VLSI chips. In *IEEE Design&Test*, 1985, s. 56–68. 18

[5] ALTET, J.; RUBIO, A.: *Thermal Testing of Integrated Circuits*. Boston, USA: Kluwer Academic Publishers, 2002, ISBN 1-4020-7076-4. 16, 30

[6] ANIS, M.; AREIBI, S.; ELMASRY, M.: Design and Optimization of Multithreshold CMOS Circuits. *IEEE Transaction on CAD*, ročník 22, č. 10, 2003: s. 1324–1342. 31

[7] BLATNÝ, J.; KOTÁSEK, Z.; HLAVIČKA, J.: RT Level Test Scheduling. *Computers and Artificial Intellignece*, ročník 16, č. 1, 1997: s. 13–29. 38

[8] BONHOMME, Y.; GIRARD, P.; GUILLER, L.; AJ.: A Gated Clock Scheme for Low Power Testing of Logic Cores. *Journal of Electronic Testing: Theory and Applications*, ročník 22, 2006: s. 89–99. 32

[9] BUSHNELL, M. L.; AGRAWAL, V. D.: *Essentials of Electronic Testing for Digital Memory & Mixed-Signal VLSI Circuits*. USA: Springer, 2000, ISBN 0-7923-7991-8, 690 s. 16, 17

[10] CHAKRABARTY, K.: Test scheduling for core-based systems using mixed-integer linear programming. In *IEEE Transactions on computer-aided design of integrated circuits and systems*, ročník 19, 2000, s. 1163–1174. 38

[11] CHAKRAVARTY, S.; DABHOLKAR, V.: Minimizing Power Dissipation in Scan Circuits During Test Application. In *Proceedings of International Workshop on Low-Power Design*, 1994, str. 20. 37, 105, 106

[12] CHOU, R. M.; SALUJA, K. K.; AGRAWAL, V. D.: Scheduling tests for VLSI systems under power constraints. In *IEEE Transactions on VLSI systems*, ročník 5, 1997, s. 175–178. 38

[13] CROUCH, A. L.: *Design for Test*. New Jersey, USA: Prentice Hall, 1999, ISBN 0-1308-4827-1, 349 s. 15

[14] DABHOLKAR, V.; CHAKRAVARTY, S.; POMERANZ, I.; AJ.: Techniques for Minimizing Power Dissipation in Scan and Combinational Circuits During Test Application. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, ročník 17, č. 12, 1998: s. 1325–1333. 36

[15] DEBJYOTI, G.; SWARUP, B.; KAUSHIK, R.: Multiple Scan Chain Design Technique for Power Reduction during Test Application in BIST. In *18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)*, 2003, s. 191–198. 26

[16] ELES, P.; KUCHCINSKI, K.; PENG, Z.: *System Synthesis with VHDL*. Dordrecht, Nizozemí: Kluwer Academic Publishers, 1998, ISBN 0-7923-8082-7, 370 s. 12

[17] GAJSKI, D.; KUHN, R.: Guest Editors' Introduction: New VLSI Tools. In *IEEE Computer*, ročník 6, 1983, s. 11–14. 12

[18] GEREZ, S. H.: *Algorithms for VLSI Design Automation*. John Willey & Sons, 1999. 14

[19] GERSTENDORFER, S.; WUNDERLICH, H. J.: Minimized Power Consumption for Scan-Based BIST. *Journal of Electronic Testing: Theory and Applications*, ročník 16, č. 3, 2000: s. 203–212. 32

[20] GIRARD, P.; GUILLER, L.; LANDRAULT, C.; AJ.: A Test Vector Inhibiting Technique for Low Energy BIST Design. In *Proceedings of the 1999 17TH IEEE VLSI Test Symposium*, 1999, ISBN 0-7695-0146-X, str. 407. 33

[21] GIRARD, P.; GUILLER, L.; LANDRAULT, C.; AJ.: A Test Vector Ordering Technique for Switching Activity Reduction During Test Operation. In *Proceedings of 9th Great Lakes Symposium on VLSI*, Washington, USA: IEEE Computer Society, 1999, ISBN 0-7695-0104-4. 37

[22] GIRARD, P.; LANDRAULT, C.; PRAVOSSOUDOVITCH, S.; AJ.: Reducing power consumption during test application by test vector ordering. In *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, IEEE Computer Society, 1998, s. 296–299. 105, 106

[23] GLOSTER, C.; BRGLEZ, F.: Partial scan selection for user-specified fault coverage. In *Proceedings of the conference on European design automation*, IEEE Computer Society Press, 1995, s. 111–116. 20

[24] GOLDSTEIN, L. H.: Controlability/Observability Analysis for Digital Circuits. In *IEEE Transactions on Circuits and Systems*, ročník 26, 1979, s. 685–693. 20

[25] HAMADA, M.; OOTAGURO, Y.: Utilizing Surplus Timing for Power Reduction. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2001, s. 89–92. 32

[26] HANSEN, M.; YALCIN, H.; HAYES, J. P.: Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design and Test*, ročník 16, 1999: s. 72–80. 97

[27] HLAVIČKA, J.: *Diagnostika a spolehlivost*. Praha: České vysoké učení technické v Praze, 1998, ISBN 8-0010-1846-6, 153 s. 15, 16

[28] HUANG, T. C.; LEE, K. J.: Reduction of Power Consumption in Scan-Based Circuits During Test Application by an Input Control Technique. *IEEE Transaction on Computer Design of Integrated Circuits and Systems*, ročník 20, č. 7, 2001: s. 911–917. 34

[29] IYENGAR, V.; CHAKRABARTY, K.: Precedence-Based, Preemptive, and Power-Constrained Test Scheduling for System-on-a-Chip. In *Proceedings of the 19th IEEE VLSI Test Symposium (VTS '01)*, 2001, s. 368–374. 38

[30] JELODAR, M. S.; AAVANI, A.: Reducing Scan Base Testing Power Using Genetic Algorithm. In *Proceedings of 11th Iranian Computer Engineering Conference*, ročník 2, 2006, s. 308–312. 103, 104

[31] KAJIHARA, S.; ISHIDA, K.; MIYASE, K.: Test Power Reduction for Full Scan Sequential Circuits by Test Vector Modification. In *Proceedings of the Second Workshop on RTL ATPG & DFT*, 2001, s. 140–145. 33

[32] KALLA, P.; CIESIELSKI, M.: A comprehensive approach to the partial scan problem using implicitstate enumeration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, ročník 21, č. 7, Jul 2002: s. 810–826. 20

[33] KOTÁSEK, Z.: RT Level Element Classification. In *Proceedings of the DDECS 97*, 1997, ISBN 80-85988-19-4, s. 41–46. 21

[34] KOTÁSEK, Z.: Uplatnění principů řiditelnosti/pozorovatelnosti při návrhu číslicových obvodů. Habilitační práce, 1999, FIT VUT v Brně, 80 stran. 21

[35] LARSSON, E.; PENG, Z.: System-on-chip test parallelization under power constraints. In *Proceedings of IEEE European Test Workshop*, 2001, s. 281–283. 38

[36] LEE, K. J.; HUANG, T. C.; CHEN, J. J.: Peak-Power Reduction for Multiple-Scan Circuits During Test Application. In *Proceedings of IEEE Asian Test Symposium*, 2000, s. 453–458. 33

[37] LONG, C.; HE, L.: Distributed Sleep Transistor Network for Power Reduction. In *Proceedings of ACM/IEEE Design Automation Conference*, 2003, s. 181–186. 31

[38] MARONGIU, A.; BENINI, L.; BARTOLINI, A.; AJ.: Analysis of Power Management Strategies for a Large-Scale SoC Platform in 65nm Technology. In *Procceedings of the 11th Euromicro Conference on Digital System Desing Architectures, Methods and Tools*, 2008, ISBN 978-0-7695-3277-6, s. 259–266. 29

[39] MENTOR GRAPHICS©: DFTAdvisor & Flextest. [Online], [rev. 2. 3. 2006], [cit. 23. 1. 2009].
URL `http://www.mentor.com/products/dft/atpg_compression/flextest/upload/DFTAdvisor_FlexTest.pdf.pdf` 20

[40] MICHELI, G. D.: *Synthesis and Optimization of Digital Circuits*. New York, USA: McGraw Hill, Inc., 1994, ISBN 0-07-016333-2. 14

[41] MÄRZ, S.: *High-Level Synthesis*, ročník 170. Boston, USA: Kluwer Academic Publishers, the kluwer international series in engineering and computer science vydání, 1992, ISBN 0-7923-9199-3, 432 s. 13

[42] NICOLICI, N.; AL-HASHIMI, B. M.: Power Conscious Test Synthesis and Scheduling for BIST RTL Data Paths. In *Proceedings of IEEE International Test Conference*, 2000, s. 662–671. 22

[43] NICOLICI, N.; AL-HASHIMI, B. M.: *Power-Constrained Testing of VLSI Circuits*. Dordrecht, Netherland: Kluwer Academic Publishers, 2003, ISBN 1-4020-7235-X, 178 s. 9, 22, 26, 27, 29, 33, 37

[44] NIRAJ, K. J.; GUPTA, S.: *Testing of Digital Systems*. Cambridge University Press, 2003, ISBN 0-5217-7356-3. 23, 34

[45] NOVAK, F.; KHALIL, M.; ROBACH, C.; AJ.: System Level Diagnostic Strategies and Tools. In *Proceedings of 4th IEEE Design and Diagnostics of Electronic Circuits and Systems*, 2001, s. 251 – 258. 21

[46] PARIKH, P. S.; ABRAMOVICI, M.: A Cost/Base Approach to Partial Scan. In *30th ACM/IEEE Design Automation Conference*, Association for Computing Machinery, 1993, s. 255 – 259. 20

[47] PARIKH, P. S.; ABRAMOVICI, M.: Testability-Based Partial Scan Analysis. *Journal of Electronic Testing: Theory and Applications*, ročník 7, č. 1, 2, 1995: s. 62–70. 20

[48] RAGHUNATHAN, A.; JHA, N. K.; DEY, S.: *High-Level Power Analysis and Optimization*. Boston, USA: Kluwer Academic Publishers, 1998, ISBN 0-7923-8073-8, 175 s. 23, 24

[49] RAVI, S.; RAGHUNATHAN, A.; CHAKRADHAR, S.: Efficient RTL Power Estimation for Large Designs. In *Proceedings of the 16th International Conference on VLSI Design*, Washington, USA: IEEE Computer Society, 2003, ISBN 0-7695-1868-0, s. 431–439. 30

[50] REEVES, C. R.; ROWE, J. E.: *Genetic algorithms: principles and perspectives: a guide to GA theory*. Kluwer Academic Publishers, 2003, ISBN 1-4020-7240-6. 47

[51] RODRIGUEZ-IRAGO, M.; ANDINA, J. J. R.; VARGAS, F.; AJ.: Dynamic Fault Test and Diagnosis in Digital Systems Using Multiple Clock Schemes and Multi-VDD Test. In *11th IEEE International On-Line Testing Symposium*, 2005, s. 281–286. 32

[52] ROSINGER, P. M.; AL-HASHIMI, B. M.; NICOLICI, N.: Power Constrained Test Scheduling Using Power Profile Manipulation. In *Proceedings of Intl. Symposium on Circuits and Systems 2001*, 2001, s. 251–254. 38

[53] ROY, K.; PRASAD, S. C.: *Low-Power CMOS VLSI Circuit Design*. USA: A Wiley-Interscience publication, 2000, ISBN 0-471-11488-X, 359 s. 9, 23, 24

[54] ROY, S.; PAL, A.: Why to Use Dual-Vt, if Single-Vt Serves the Purpose Better under Process Parameter Variations? In *Proceedings of 11th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, 2008, ISBN 978-0-7695-3277-6, s. 282–289. 31

[55] RŮŽIČKA, R.: *Formální přístup k analýze testovatelnosti číslicového obvodu na úrovni RT*. Dizertační práce, FIT VUT v Brně, 2002. 20, 50

[56] SANKARALINGAM, R.; ORUGANTI, R. R.; TOUBA, N. A.: Static Compaction Techniques to Control Scan Vector Power Dissipation. In *Proceedings of IEEE VLSI Test Symposium*, 2000, s. 35–40. 27, 33

[57] SANKARALINGAM, R.; POUYA, B.; TOUBA, N. A.: Reducing Power Dissipation During Test Using Scan Chain Disable. In *Proceedings of the IEEE VLSI Test Symposium*, 2001, s. 319–324. 33

[58] SATHANUR, A.; PULLINI, A.; BENINI, L.; AJ.: Timing-Driven Row-Based Power Gating. In *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design*, 2007, s. 104–109. 31

[59] SCHMITZ, M. T.; AL-HASHIMI, B. M.; ELES, P.: *System-Level Design Techniques for Energy-Efficient Embedded Systems*. Boston, USA: Kluwer Academic Publishers, 2004, ISBN 1-4020-7750-5, 211 s. 18, 32

[60] SCHUELE, T.; STROELE, A.: Test Scheduling for Minimal Energy Consumption under Power Constraints. In *Proceedings of the 19th IEEE VLSI Test Symposium*, Washington, USA: IEEE Computer Society, 2001, ISBN 1093-0167, s. 312–318. 38

[61] SHANNON, C. E.: Communication in the Presence of Noise. *Proc. Institute of Radio Engineers*, ročník 37, č. 1, 1949: s. 10–21, reprint as classic paper in: Proc. IEEE, Vol. 86, No. 2, (Feb 1998). 30

[62] SHIVELEY, R.: Dual-Core Intel©Itanium©2 Processors Deliver Unbeatable Flexibility and Performance to the Enterprise. *Technology@Intel Magazine*, August 2006. 16

[63] STRNADEL, J.: *Analýza a zlepšení testovatelnosti číslicového obvodu na úrovni meziregistrových přenosů*. Dizertační práce, FIT VUT v Brně, 2004. 21

[64] STROUD, C. E.: *A Designer's guide: Built-In Self-Test*. Boston USA: Kluwer Academic Publishers, 2002, ISBN 1-4020-7050-0. 17

[65] SU, C.-P.; WU, C.-W.: Graph-based power constrained test scheduling for SOC. In *Proceedings of IEEE design and diagnostics of electronic circuits and system workshop*, 2002, ISBN 0-2142-094-4, s. 61–68. 38

[66] THOMPSON, S.; PACKAN, P.; BOHR, M.: MOS Scaling: Transistor Challanges for the 21st Century. *Intel Technology Journal*, ročník 19, 1998. 29

[67] USAMI, K.; HOROVITZ, M.: Clustered Voltage Scaling Techniques for Low-Power Design. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 1995. 32

[68] VEENDRICK, H. J. M.: Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid State Circuits*, ročník 19, 1984: s. 468–473. 29

[69] VRANKEN, H.; WAAYERS, T.; FLEURY, H.; AJ.: Enhanced Reduced-Pin-Count Test for Full-Scan Design. In *Proceedings of the IEEE International Test Conference*, 2001, s. 738–747. 32

[70] WAKERLY, J. F.: *Digital Design, Principles and Practices*. New Jersey, USA: Prentice Hall, 2000, ISBN 0-1376-9191-2. 14

[71] WANG, S.; GUPTA, S. K.: ATPG for Heat Dissipation Minimization During Test Application. *IEEE Transactions on Computers*, ročník 47, č. 2, 1998: s. 256–262. 33

[72] WHETSEL, L.: Adapting Scan Architectures for Low Power Operation. In *Proceedings of IEEE International Test Conference*, 2001, s. 652–659. 33

[73] WILLIAMS, M. J.; ANGEL, J. B.: Enhancing Testability of Large-Scale integrated Circuits via Test points and Additional Logic. *IEEE Transactions on Computers*, ročník 1, 1973: s. 46–60. 20

[74] WILLIAMS, T. W.: EDA to the Rescue of the Silicon Roadmap. In *Proceedings of the 38th International Symposium on Multiple Valued Logic*, 2008, ISBN 978-0-7695-3155-7, s. 1–1. 31

[75] XU, L.; SUN, Y.; CHEN, H.: Scan Array Solution for Testing Power and Testing Time. In *Proceedings of IEEE International Test Conference*, 2001, s. 652–659. 33

[76] ŠKARVADA, J.: BEAST Homepage. [Online], [rev. 1. 12. 2008], [cit. 1. 12. 2008]. URL `http://www.fit.vutbr.cz/~skarvada/beast/` 42, 90, 92, 93, 95, 96

[77] ŠKARVADA, J.: Test Scheduling for SOC under Power Constraints. In *Proceedings of the 2006 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Praha: Vydavatelství ČVUT, 2006, ISBN 1-4244-0184-4, s. 91–93. 38

[78] ŠKARVADA, J.; HERRMAN, T.; KOTÁSEK, Z.: Testability Analysis Based on the Identification of Testable Blocks with Predefined Properties. In *Proceedings of 10th EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN Architectures*, Lübeck, Německo: IEEE Computer Society, 2007, ISBN 0-7695-2978-X, s. 611–618. 75, 112

Seznam obrázků

Seznam použitých zkratek a symbolů

Symboly v latince

B	Množina představující allelu
C	Relace definující metalické spoje v obvodu
CI	Množina všech řidicích a synchronizačních bran
D	Množina definující obor hodnot signálu
E	Množina obvodových prvků
f	Obecná funkce
FU	Množina všech funkčních prvků
IN	Množina všech datových vstupních bran
in	Relace přiřazující vstupní brány jednotlivým obvodovým prvkům
in_ord	Relace přiřazující vstupním branám identifikační čísla z množiny přirozených čísel
$inscan\_ord$	Relace přiřazující datovým vstupním branám každého scan registru identifikační čísla z množiny přirozených čísel
l	Konstanta určující délku chromozómu
MT	Množina definující příkonové metriky
MX	Množina všech multiplexorů
N	Množina individuí
OUT	Množina všech datových výstupních bran
out	Relace přiřazující výstupní brány jednotlivým obvodovým prvkům
out_ord	Relace přiřazující výstupním branám identifikační čísla z množiny přirozených čísel
$outscan\_ord$	Relace přiřazující datovým výstupním branám každého scan registru identifikační čísla z množiny přirozených čísel

P	Množina všech bran všech obvodových prvků
PI	Množina všech primárních vstupních bran
pin_ord	Relace přiřazující primárním vstupním branám identifikační čísla z množiny přirozených čísel
PO	Množina všech primárních výstupních bran
R	Množina všech registrů
S	Množina všech možných řešení zkoumané úlohy
s	Výběrový operátor GA pro výběr jedinců z populace dle jejich kvality
SC	Množina všech scan registrů
SR_i	Jednotlivé scan řetězce
SRS	Topologie scan řetězců
SV_i	Jednotlivé scan vektory
SVS	Posloupnost scan vektorů
TV_i	Jednotlivé testovací vektory
TVS	Testové cykly

Symboly řecké

Δ	Relace přiřazující každému chromozómu řešení z prostoru řešení
η	Relace kvantifikuje příkonovou metriku obvodových prvků při přechodu mezi jednotlivými stavy
λ	Konstanta určující velikost populace GA
ν	Přechodová funkce obvodových prvků
Ω	Množina definující genetické operátory
Φ	Fitness funkce
τ	Relace definující ukončující podmínku GA
Ξ	Přechodová funkce mezi jednotlivými generacemi GA
ζ	Relace definující příkon scan registrů v jednotlivých příkonových metrikách

Zkratky

ALAP	As Last As Possible – nastavení testovacích vektorů co nejpozději
ASAP	As Soon As Possible – nastavení testovacích vektorů co nejdříve
ASIC	Application Specific Integrated Circuit – aplikačně specifický integrovaný obvod

ATE	Automatic Test Equipment – zařízení pro samočinné vnější testování
BIST	Built-In Self-Test – vestavěný test
BPIC	Best Primary Input Change time – nastavení testovacích vektorů v nejvýhodnější dobu
CUA	Circuit under Analysis - analyzovaný číslicový obvod
DfT	Design for Testability - návrh s ohledem na snadnou testovatelnost
GA	Genetický algoritmus
GTV	generátor testovacích vektorů
HDL	Hardware Description Language – jazyk pro popis hardware
IO	integrovaný obvod
LFSR	Linear Feedback Shift Register – lineární posuvný registr se zpětnou vazbou
LPRZV	lineární posuvný registr se zpětnou vazbou
NTC	Number of Transition Count - počet překlopení $0 \rightarrow 1$, $1 \rightarrow 0$
PA	příznakový analyzátor
POST	Power-On Self-Test – test po zapnutí zařízení
RTL	Register Transfer Level – úroveň meziregistrových přenosů
SA	Signature Analyser – příznakový analyzátor
SoC	System on a Chip – systém na jednom čipu
TPC	Test per Clock - aplikace testovacích vektorů po jednotlivých synchronizačních pulzech
TPG	Test Pattern Generator – generátor testovacích vektorů
TPS	Test per Scan - aplikace testovacích vektorů po jednotlivých cyklech řetězce scan
TTM	Time to Market – čas pro uvedení na trh
VLSI	Very Large Scale Integrated - technologie pro výrobu obvodů s velmi vysokým stupněm integrace
WNTC	Weighted Number of Transition Count - počet vážených překlopení $0 \rightarrow 1$, $1 \rightarrow 0$
WSA	Weighted Switching Activity - vážená přepínací aktivita

Rejstřík