České vysoké učení technické v Praze

Fakulta elektrotechnická

Diplomová práce

Návrh generátoru pseudonáhodných vektorů pro vestavěnou diagnostiku

Bc. Pavel Galaktionov

Vedoucí práce: Ing. Petr Fišer, PhD.

Studijní program: Informatika a výpočetní technika

prosinec 2011

Poděkování

Rád bych na tomto místě poděkoval zejména svému vedoucímu diplomové práce Ing. Petru Fišerovi za odborné vedení, pomoc, konzultace a za poskytnuté materiály. Dík patří také mým blízkým, kamarádům a mé přítelkyni za psychickou podporu, kterou mi během studia i realizace této diplomové práce poskytovali.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu. Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb. , o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 18. prosince 2011

…….

Abstract

This work deals with the implementation of selected algorithms for the implementation of the generators of pseudorandom vectors (Pseudo-Random Pattern Generators - PRPGs) for built-in diagnostics (Built-In Self-Test - BIST) for selected types of circuits. On each circuit there is known number of detectable and undetectable faults.

The aim is to design and implement algorithms implementing such ~~a~~ test vector generators. These vectors are used to detect all detectable faults. Input of each of such an algorithm is a whole range of parameters. By the necessary number of vectors I mean the minimum number of vectors that detect all detectable faults.

Another task is to find the settings so that a compromise is found between the number of vectors, the algorithm runtime and memory size that is needed. Still, there is a condition that that all detectable faults must be detected

Finally, the results are compared different variants of algorithms.

Anotace

Tato práce se zabývá implementací vybraných algoritmů pro realizaci návrhu generátorů pseudonáhodných vektorů (Pseudo-Random Pattern Generators - PRPGs) pro vestavěnou diagnostiku (Built-In Self-Test - BIST) na vybraných typech obvodů. O každém obvodu jsou známy počty detekovatelných a nedetekovatelných poruch.

Cílem je navrhnout a implementovat algoritmy realizující takové generátory testovacích vektorů tak, aby vektory detekovaly všechny detekovatelné poruchy. Vstupem každého takového takového algoritmu je celá řada parametrů.

Nutným počtem vektorů rozumím takový minimální počet vektorů, který detekuje všechny detekovatelné poruchy.

Dalším úkolem je tedy najít takové nastavení parametrů tak, aby byl nalezen určitý kompromis mezi počtem vektorů, dobou běhu algoritmu a velikostí paměti, která je zapotřebí. Stále ale platí podmínka, že musí být detekovány všechny detekovatelné poruchy.

Nakonec jsou porovnány výsledky různých variant algoritmů.

Obsah

Seznam obrázků
Seznam tabulek
Seznam algoritmů
Seznam použitých zkratek

Obsah

1 Úvod 1
1.1 Úvod 1
1.2 Zadaný problém 1
1.3 Cíle práce 1
1.4 Struktura práce 1

2. Detailní popis problému 2
2.1 Detailní popis problému 2

3. Analýza a testování 3
3.1 Schema testování obvodu 3
3.2 Generátory testovacích vektorů 3
3.3 Základní typy generátorů 3
3.3.1 Lineární zpětnovazební posuvní registr (LZPR) 3
3.3.1.1 Generující polynom 4
3.3.2 Celulární automat (CA) 4
3.4. Čistě deterministické testování (ČDT) 5
3.4.1 Průběh a schema ČDT 5
3.4.2 Výsledky ČDT pro vybrané obvody [11], [12] 6
3.5 Čistě pseudonáhodné testování (ČPT) 9
3.5.1 Vlastní implementace algoritmu ČPT 9
3.5.2 Generování LZPR sekvence pomocí ČPT 10

4. Reseeding 14
4.1 Vlastní implementace jednopolynomiálního reseedingu (var. 1.) 14
4.2 Porovnání výsledků ČPT s výsledky jednopolynomiálního reseedingu (var. 1.) 15
4.3 Porovnání počtu detekovaných poruch při ČPT a reseedingu (var. 1.) 18
4.4 Vlastní implementace jednopolynomiálního reseedingu (var. 2.) 19
4.4.1 Analýza a výsledky jednopolynomiálního reseedingu var. 2. 20
4.5 Vlastní implementace jednopolynomiálního reseedingu (var. 3.) 23
4.5.1 Analýza a výsledky jednopolynomiálního reseedingu (var. 3.) 24
4.5.2 Výsledky měření pro proměnnou délku sekvence PR vektorů 24
4.5.2.1 Výsledky měření pro obvod s838 25
4.5.2.2 Výsledky měření pro obvod s713 27
4.5.2.3 Výsledky měření pro obvod s420 29
4.5.2.4 Výsledky měření pro obvod c880 31

5. Porovnání počtu seedů LZPR a CA (prav. 60) 34
5.1 Výsledky měření pro obvod s838 34
5.2 Výsledky měření pro obvod s713 36
5.3 Výsledky měření pro obvod s420 38
5.4 Výsledky měření pro obvod c880 40

5.5 Srovnání výsledků reseedingu s výsledky MPLFSR..... 42
5.6 Shrnutí..... 42

6. Implementace 43
6.1 Program..... 43
6.2 Vstupní data.....43
6.3 Výstupní data.....43
6.4 Urychlení testování.....44

7. Závěr 45

Seznam obrázků

Schema 2.1 Porovnání architektur z hlediska použitých testovacích vektorů....2
Schema 3.1 Schema testování obvodu 3
Obrázek 3.1 Schema lineárního zpětnovazebního registru 4
Obrázek 3.2 Schema celulárního automatu 4
Schema 3.2 Obecné schema čistě deterministického testování 5
Graf 3.1 Vývoj počtu detekovaných poruch při ČDT (obvod c499) 6
Graf 3.2 Vývoj počtu detekovaných poruch při ČDT (obvod c880) 6
Graf 3.3 Vývoj počtu detekovaných poruch při ČDT (obvod c1908) 7
Graf 3.4 Vývoj počtu detekovaných poruch při ČDT (obvod c1355) 7
Graf 3.5 Vývoj počtu detekovaných poruch při ČD (obvod c5315) 7
Graf 3.6 Vývoj počtu detekovaných poruch při ČD (obvod s838) 8
Schema 3.3 Schema algoritmu čistě pseudonáhodného testování 9
Graf 3.7 Zobrazení vývoje počtu detekovaných poruch při pseudonáhodném testování (obvody c1908 a c880) 10
Graf 3.8 Zobrazení vývoje počtu detekovaných poruch při pseudonáhodném testování (obvody c1908 a c880) 11
Graf 3.9 Rozptyl hodnot počtu vektorů pro detekci všech detekovat poruch (obvod c499) 12
Graf 3.10 Rozptyl hodnot počtu vektorů pro detekci všech detekovat poruch (obvod c880) 12
Graf 3.11 Rozptyl hodnot počtu vektorů pro detekci všech detekovat poruch (obvod c1908) 13
Graf 3.12 Rozptyl hodnot počtu vektorů pro detekci všech detekovat poruch (obvod c1355) 13
Graf 3.13 Rozptyl hodnot počtu vektorů pro detekci všech detekovat poruch (obvod c5315) 13
Schema 4.1 Schema algoritmu jednopolynomiálního reseedingu (var 1) 15
Graf 4.1 Porovnání počtu vektorů pro detekci všech detekovat poruch (obvod c499) 16
Graf 4.2 Porovnání počtu vektorů pro detekci všech detekovat poruch (obvod c880) 16
Graf 4.3 Porovnání počtu vektorů pro detekci všech detekovat poruch (obvod c1908) 16
Graf 4.4 Porovnání počtu vektorů pro detekci všech detekovat poruch (obvod c1355) 17
Graf 4.5 Porovnání počtu vektorů pro detekci všech detekovat poruch (obvod c5315) 17
Graf 4.6 Porovnání počtu detekovaných poruch pro dané počty vektorů získané pomocí ČPT s počty vektorů reseedingu (var 1) (obvod s838) 18
Graf 4.7 Porovnání počtu detekovaných poruch pro dané počty vektorů získané pomocí ČPT s počty vektorů reseedingu (var 1) (obvod s713) 18
Schema 4.2 Schema algoritmu jednopolynomiálního reseedingu (var 2) 19
Graf 4.8 Nárůst počtu detekovaných poruch v inicializační fázi pro vybrané obvody 21
Graf 4.9 Skutečný průběh detekce poruch při reseedingu var 2 pro dvě různá měření (obvod s838) 22
Graf 4.10 Rozptyl počtu detekovaných poruch pro 100 měření reseedingu (obvod s838) 22
Graf 4.11 Rozptyl počtu detekovaných poruch pro 100 měření reseedingu (obvod s713) 22
Schema 4.3 Schema algoritmu jednopolynomiálního reseedingu (var 3) 24
Graf 4.12 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s838) 25
Graf 4.13 Porovnání míst reseedingu dle délky sekvence (obvod s838) 25
Graf 4.14 Vývoj počtu seedů dle délky sekvence (obvod s838) 25
Graf 4.15 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s838) 26
Graf 4.16 Porovnání míst reseedingu dle délky sekvence (obvod s838) 26
Graf 4.17 Vývoj počtu seedů dle délky sekvence (obvod s838) 26
Graf 4.18 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s713) 27
Graf 4.19 Porovnání míst reseedingu dle délky sekvence (obvod s713) 27
Graf 4.20 Vývoj počtu seedů dle délky sekvence (obvod s713) 27
Graf 4.21 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s713) 28
Graf 4.22 Porovnání míst reseedingu dle délky sekvence (obvod s713) 28
Graf 4.23 Vývoj počtu seedů dle délky sekvence (obvod s713) 28
Graf 4.24 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s420) 29

Graf 4.25 Porovnání míst reseedingu dle délky sekvence (obvod s420) 29
Graf 4.26 Vývoj počtu seedů dle délky sekvence (obvod s420) 29
Graf 4.27 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s420) 30
Graf 4.28 Porovnání míst reseedingu dle délky sekvence (obvod s420) 30
Graf 4.29 Vývoj počtu seedů dle délky sekvence (obvod s420) 30
Graf 4.30 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod c880) 31
Graf 4.31 Porovnání míst reseedingu dle délky sekvence (obvod c880) 31
Graf 4.32 Vývoj počtu seedů dle délky sekvence (obvod c880) 31
Graf 4.33 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod c880) 32
Graf 4.34 Porovnání míst reseedingu dle délky sekvence (obvod c880) 32
Graf 4.35 Vývoj počtu seedů dle délky sekvence (obvod c880) 32
Graf 5.1 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 51) (obvod s838) 34
Graf 5.2 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 52) (obvod s838) 35
Graf 5.3 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 53) (obvod s838) 35
Graf 5.4 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 54) (obvod s838) 35
Graf 5.5 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 55) (obvod s713) 36
Graf 5.6 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 56) (obvod s713) 37
Graf 5.7 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 57) (obvod s713) 37
Graf 5.8 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 58) (obvod s713) 37
Graf 5.9 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 59) (obvod s420) 38
Graf 5.10 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 510) (obvod s420) 39
Graf 5.11 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 511) (obvod s420) 39
Graf 5.12 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 512) (obvod s420) 39
Graf 5.13 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 513) (obvod c880) 40
Graf 5.14 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 514) (obvod c880) 41
Graf 5.15 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 515) (obvod c880) 41
Graf 5.16 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 516) (obvod c880) 41
Schema 6.1 Schema algoritmu vyhodnocování počtu poruch v sekvenci 44

Seznam tabulek

Tab. 3.1 Vybraná pravidla celulárních automatů 4
Tab. 3.2 Počty deterministických vektorů 6
Tab. 3.3 Přehled počtu nutných vektorů pro vybrané obvody 11
Tab. 3.4 Srovnání výsledků uváděných v literatuře s výsledky naměřenými....12
Tab. 4.1 Srovnání počtu vektorů pro jednotlivé obvody při reseedingu (var. 1.), ČPT a ČDT ...17
Tab. 4.2 Přehled počtu detekovaných poruch v inicializační fázi pro vybrané obvody 21
Tab. 5.1 až 5.4 počet seedů pro sekvenci LZPR a CA (60), obvod s838 34
Tab. 5.5 až 5.8 počet seedů pro sekvenci LZPR a CA (60) (obvod s713) 36
Tab. 5.9 až 5.12 počet seedů pro sekvenci LZPR a CA (60) (obvod s420) 38
Tab. 5.13 až 5.16 počet seedů pro sekvenci LZPR a CA (60) (obvod c880) 40
Tab. 5.17 Srovnání výsledků reseedingu s výsledky MPLFSR 42

Seznam algoritmů

3.4. Čistě deterministické testování (ČDT) 5
3.5 Čistě pseudonáhodné testování (ČPT) 9
4.1 Vlastní implementace jednopolynomiálního reseedingu (var. 1.) 14
4.4 Vlastní implementace jednopolynomiálního reseedingu (var. 2.) 19
4.5 Vlastní implementace jednopolynomiálního reseedingu (var. 3.)....23

Seznam použitých zkratek

VLSI zařízení s vysokou hustotou integrace (very large scale integration)
ATE přístroj pro automatické testování (automated test equipment)
BIST vestavěný test (built-in self-test)
PRPG generátoru pseudonáhodných vektorů (pseudo-random pattern generator)
LZPR lineární zpětnovazební posuvný registr
LFSR linear feedback shift register
MPLFSR multiple polynomial linear feedback shift register
CA celulární automat (cellular automaton)
ČPT čistě pseudonáhodné testování
ČDT čistě deterministické testování
PR pseudonáhodné (pseudo-random)

1. Úvod

1.1 Úvod

Ze složitosti současných VLSI (Very Large Scale Integration) zařízení, což jsou zařízení s vysokou hustotou integrace, kde na jeden čip připadá řádově několik desítek tisíc prvků, vyplývá, že jednotlivé čipy již nejsou v rozumně krátké době otestovatelné standardními testy. Ty jsou provedeny při výrobě daného čipu pomocí automatizovaného zařízení ATE (Automated Test Equipment) a testovacích vektorů.

Snahou je tedy zkrátit dobu testování, a proto jsou hledány nové způsoby a techniky, a jednou z takovýchto technik je BIST (Built-In Self-Test). Pomocí dané techniky je obvod schopen otestovat sám sebe bez nutnosti použití dalších testů, či jiných prostředků. BIST napomáhá snížit počet vektorů a tím i zkrátit dobu nutnou k otestování čipu. Navíc paměťové nároky na takovýto test jsou menší.

Technika BIST také využívá testovacích vektorů. Ty jsou získané buď deterministickými metodami a mluvíme pak o deterministických testovacích vektorech, anebo, a to ve většině případů, pomocí generátoru pseudonáhodných (PR) vektorů. Takový generátor produkuje testovací vektory, jimiž se detekují takzvané lehce detekovatelné poruchy, které tvoří přes 90% všech poruch. Zbývající procenta poruch je nutno detekovat jinak.

Například pro obvod s838, který je nejhůře testovatelným obvodem v této práci, je potřeba k detekci všech detekovatelných poruch přes sto milionů PR vektorů.

1.2 Zadaný problém

Máme sadu vybraných testovaných obvodů různé složitosti a o každém dílčím obvodu víme, krom jiných údajů i to, kolik má detekovatelných a kolik nedetekovatelných poruch. Problémem je najít způsob testování takový, aby pro každý obvod bylo detekováno co nejvíce detekovatelných poruch.

1.3 Cíle práce

Cílem této práce je nastudovat možné způsoby návrhu generátoru pseudonáhodných vektorů (PRPG) pro vestavěnou diagnostiku (BIST) a provést jejich rešerši.

Dále pak vytvořit program pro simulaci běhu zvolených PRPG a s jeho pomocí vyhodnotit jejich vlastnosti (pokrytí poruch, časová a prostorová složitost, počet nutných testovacích vektorů).

A konečně pro dekompresor daného obvodu použít algoritmus pro kompresi testovacích vektorů. Dosažené výsledky zobrazit graficky.

1.4 Struktura práce

V první kapitole je lehký úvod do problematiky, zadání problému a cíle práce.

V druhé je detailněji rozebrán daný problém.

Třetí kapitola obsahuje analýzu, testování a prezentaci výsledků deterministického a čistě pseudonáhodného testování.

Čtvrtá kapitola se zabývá reseedingem a několika jeho modifikacemi. Obsahuje též analýzu a výsledky testování.

V páté je porovnání počtu reseedingů pro sekvence vzniklé kroky LZPR a za použití celulárního automatu CA (pravidlo 60).

Šestá kapitola stručně popisuje vlastní implementaci mého programu a struktury dat.

A konečně sedmá, poslední, je závěr a zhodnocení práce.

2. Detailní popis problému

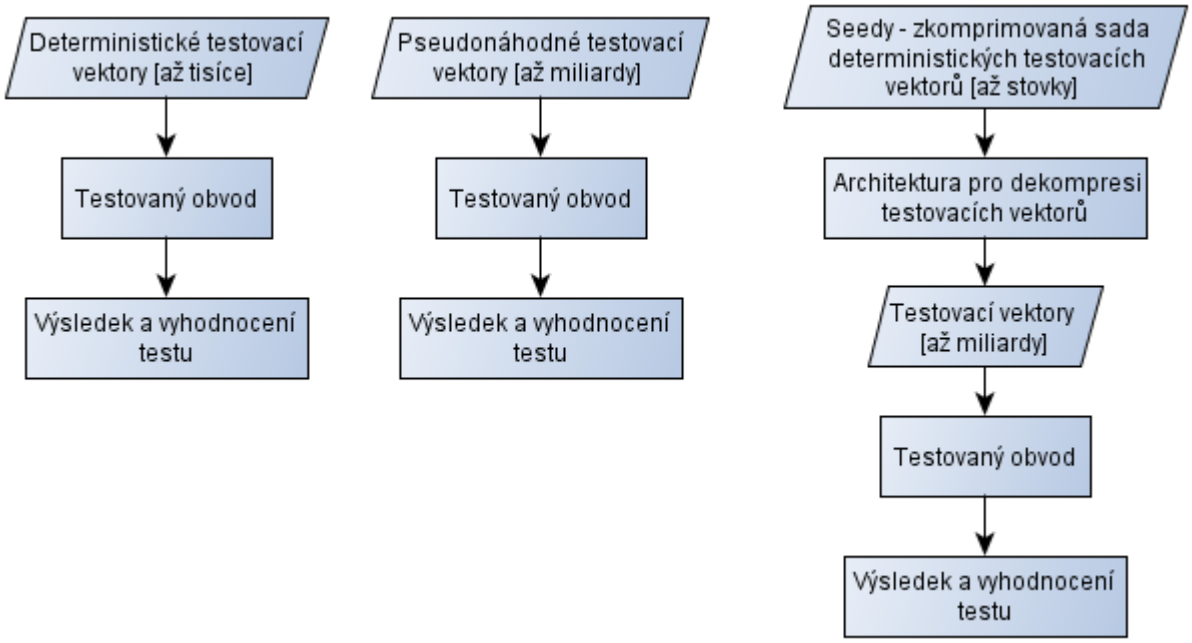
V této kapitole je detailněji specifikován daný problém a prezentován rozdíl mezi třemi architekturami, které se liší dle typu použitých testovacích vektorů.

2.1 Detailní popis problému

Máme tedy různé obvody a v každém takovémto obvodu se mohou vyskytnou poruchy. Ze specifikace obvodu víme, kolik má detekovatelných a kolik nedetekovatelných poruch. K detekci poruch se používají testovací vektory. U obvodů snadno PR testovatelných stačí desítky až tisíce PR vektorů, naproti tomu u obvodů velmi obtížně otestovatelných na PR poruchy jsou zapotřebí stovky milionů až miliardy PR vektorů.

Používáme-li pouze PR vektory, mluvíme pak o čistě pseudonáhodném testování (ČPT). Sice nepotřebujeme žádný prostor pro uchovávání vektorů, ale doba testu může být neúnosně dlouhá.

Opačným extrémem je použití čistě deterministického testu (ČDT). V tomto případě sice docílíme krátké doby testování, ale musíme si uchovávat velký počet deterministických testovacích vektorů a tedy i velký objem dat. Abychom tento objem snížili, využíváme kompresi dat. Díky ní nemusíme uchovávat všechny deterministické vektory, ale jen seedy (podkapitola 3.3.1.1). Zbývající "chybějící" deterministické vektory jsou nahrazeny PR vektory, které vygenerujeme pomocí LZPR (podkapitola 3.3.1), nebo pomocí CA (podkapitola 3.3.2). Přístup, v němž jsem využil obě dvě možnosti, je reseeding, který je stěžejním prvkem této práce, a jež je dopodrobna popsán v samostatné kapitole 4.



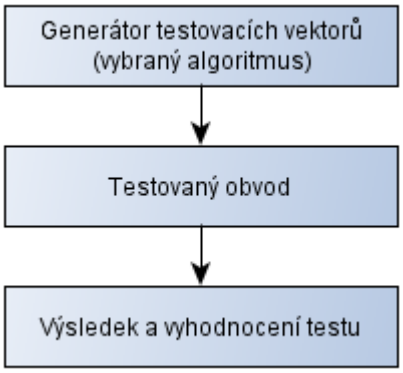
Schema 2.1 Porovnání architektur z hlediska použitých testovacích vektorů

3. Analýza a testování

V této kapitole je čtenář postupně seznámen se schematem testování obvodu, dále pak se základními pojmy, s analýzou a popisem algoritmů a s výsledky testování. Hlavně z důvodu lepší čitelnosti textu práce nejsou měření a výsledky prezentovány v samostatné kapitole, ale jsou uvedeny ihned za popisem každého algoritmu.

3.1 Schema testování obvodu

Vzhledem k tomu, že všechny algoritmy zmiňované v této práci, slouží vlastně jako generátory testovacích vektorů pro daný obvod, můžeme celý proces testovaní popsat následujícím schematem.



Schema 3.1 Schema testování obvodu

3.2 Generátory testovacích vektorů

Generátory testovacích vektorů jsou takové struktury, které při své činnosti na výstupu produkují testovací vektory. Pod pojmem testovací vektor si lze představit libovolný řetězec určité délky, který je tvořen posloupností nul a jedniček, eventuálně ještě uvažujeme X, který značí neurčený stav. Může to být jak jednička, tak nula. Převedeno do výrazů booleovy algebry, *1 = true (pravda), 0 = false (nepravda)*. Ve své práci se zabývám syntézou generátorů testovacích vektorů, které detekují všechny detekovatelné poruchy. Respektive běh pokročilého algoritmu generátoru testovacích vektorů končí, až když jsou detekovány všechny detekovatelné poruchy.

3.3 Základní typy generátorů

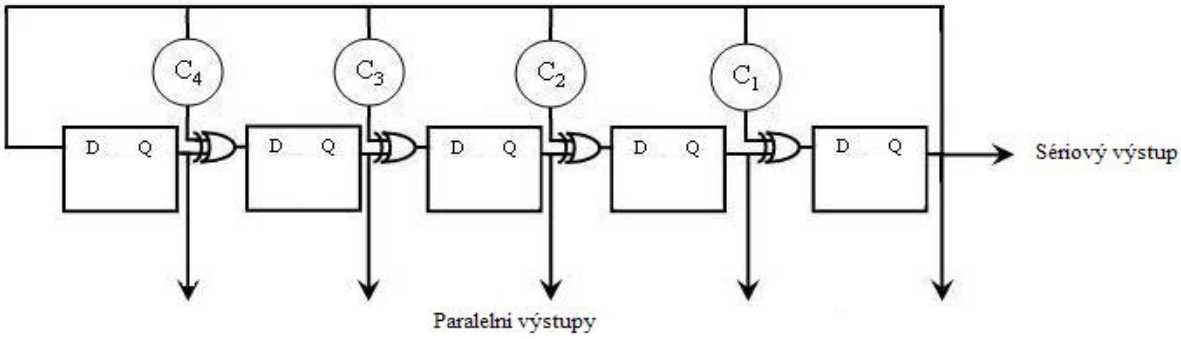
Generátor pseudonáhodných vektorů je jednoduchý obvod, který generuje kódová slova dle zadaného generujícího polynomu, který je blíže popsán v odstavci 3.3.1.1 Mezi základní typy generátorů testovacích vektorů řadíme Lineární zpětnovazební posuvní registr (LZPR), viz. 3.3.1 a celulární automat (CA), který je stručně popsán v odstavci 3.3.2

3.3.1 Lineární zpětnovazební posuvní registr (LZPR)

Je nejběžněji používaný generátor. Jeho české pojmenování odpovídá anglickému ekvivalentu LFSR (Linear feedback shift register) a pod tímto anglickým pojmenováním je také snáze vyhledatelný. Jedná se o lineární sekvenční obvod, který se skládá z klopných obvodů typu “D“ a obvodů typu XOR generujících kódové slovo cyklického kódu. Kódovým slovem zde opět rozumíme binární vektor. U konkrétního cyklického kódu již musíme nutně rozlišovat kódová slova přípustná a kódová slova nepřípustná. Přípustná kódová slova jsou ta, která splňují podmínku definující cyklický kód. Naproti tomu kódová slova nepřípustná tuto podmínku nesplňují. Více v [2] a [3]. Schema LZPR je uvedeno na obrázku 3.1.

3.3.1.1 Generující polynom

Sekvence kódových slov získaná pomocí LZPR může být popsána generujícím polynomem zapsaným v obecném tvaru $g(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_1x^1 + 1$, kde *n* je počet paralelních výstupů. Generující polynom pro obrázek 3.1 by odpovídal zápisu $g(x) = x^5 + c_4x^4 + c_3x^3 + + c_2x^2 + c_1x + 1$. Počáteční nastavení registru, čili nastavení klopných obvodů se anglicky nazývá *seed*.



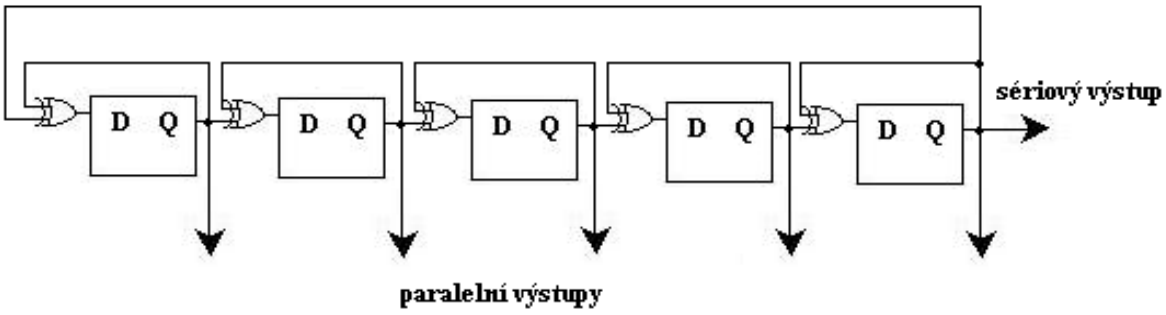
Obrázek 3.1 Schema lineárního zpětnovazebního registru

Koeficienty $C_4 - C_1$ odpovídají koeficientům v generujícím polynomu a vyjadřují, zda v daném místě existuje (1), či neexistuje (0) spojení ze zpětné vazby do příslušného XORu.

3.3.2 Celulární automat (CA)

Je sekvenční obvod podobný obvodu LZPR, uvedenému na obr. 3.1. Jeho perioda je ovšem často oproti LZPR kratší, nicméně kódová slova jím generovaná se jeví v mnoha případech vhodnější. Ve své práci používám oba generátory, jak LZPR, tak CA a proto je nutné uvést, že u CA se uvádí i pravidla pro to, jak se po každém taktu určují hodnoty v každé buňce (klopném obvodu typu D) automatu. Z toho vyplývá i to, jak budou vznikat testovací vektory. Já používal pravidlo 60, viz. Tab 3.1 níže. Více v [3] a [4].

Pro úplnost dodávám, že pokud se na všechny buňky CA uplatňuje stejné pravidlo, jedná se o homogenní CA, v opačném případě se jedná o hybridní CA.



Obrázek 3.2 Schema celulárního automatu

pravidlo	určení hodnoty v dané buňce
60	xor hodnoty buňky a buňky od ní nalevo
90	xor hodnot okolních buněk
102	xor hodnoty buňky a buňky od ní napravo
150	xor hodnoty buňky s hodnotami okolních buněk
170	hodnota buňky napravo
204	hodnota dané buňky
240	hodnota buňky nalevo

Tab. 3.1 Vybraná pravidla celulárních automatů

3.4. Čistě deterministické testování (ČDT)

Jedná se v podstatě o extrémní případ testování. Pomocí ATPG Atalanta [8] vygeneruji pro daný obvod množinu deterministických testovacích vektorů, která splňuje následující dva požadavky. Prvním požadavkem je, že množina vektorů je minimální. Minimální v tom smyslu, že pokud bychom z ní odebrali libovolný testovací vektor, nepodaří se nám detekovat všechny detekovatelné poruchy.

Z předchozího požadavku tedy vyplývá požadavek následující a tím je právě to, že pomocí této sady (množiny) testovacích vektorů musíme být schopni detekovat všechny detekovatelné poruchy.

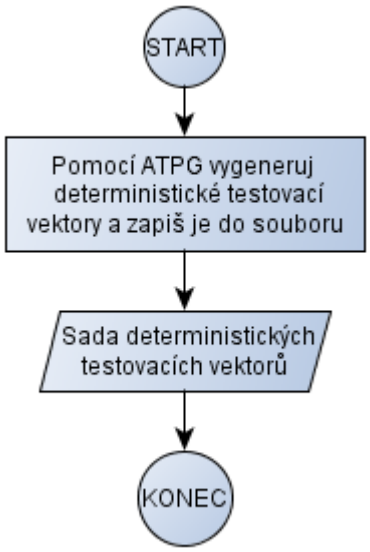
Výhodou tohoto testování je krátká doba testu, naopak nevýhodou skutečnost, že všechny deterministické vektory musíme někde uchovávat. Buď přímo na vlastním čipu, nebo na externím úložišti. Objem takovýchto uchovávaných dat může být ale značný.

3.4.1 Průběh a schema ČDT

Chceme-li vědět, jaký byl vývoj počtu detekovaných poruch při ČDT, budeme postupovat dle následujícího schematu. To se skládá z několika kroků, v nichž se cyklicky opakuje načtení deterministického vektoru (v mém případě ze souboru) a simulace testu. Testování končí, když jsme odsimulovali test nad každým vektorem. Tím ovšem máme zaručeno i to, že byly detekovány všechny detekovatelné poruchy.

Slovně lze průběh ČDT (po nastavení parametrů) popsat následovně:

1. Pro daný obvod vygeneruj pomocí ATPG deterministické vektory a ty ulož do souboru
2. Pro každý vektor odsimuluj test a výsledky testu (počet detekovaných poruch a report z testu) zapiš do souborů
4. Skonči, pokud byl test simulován pro všechny vektory (mám zaručeno, že byly detekovány všechny detekovatelné poruchy)



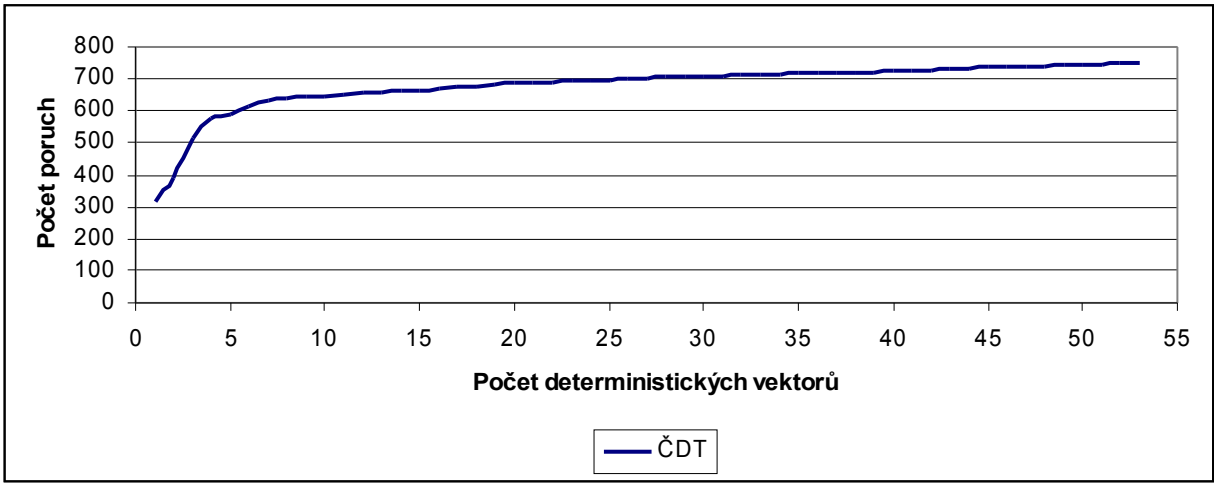
Schema 3.2 Obecné schema čistě deterministického testování

3.4.2 Výsledky ČDT pro vybrané obvody [11], [12]

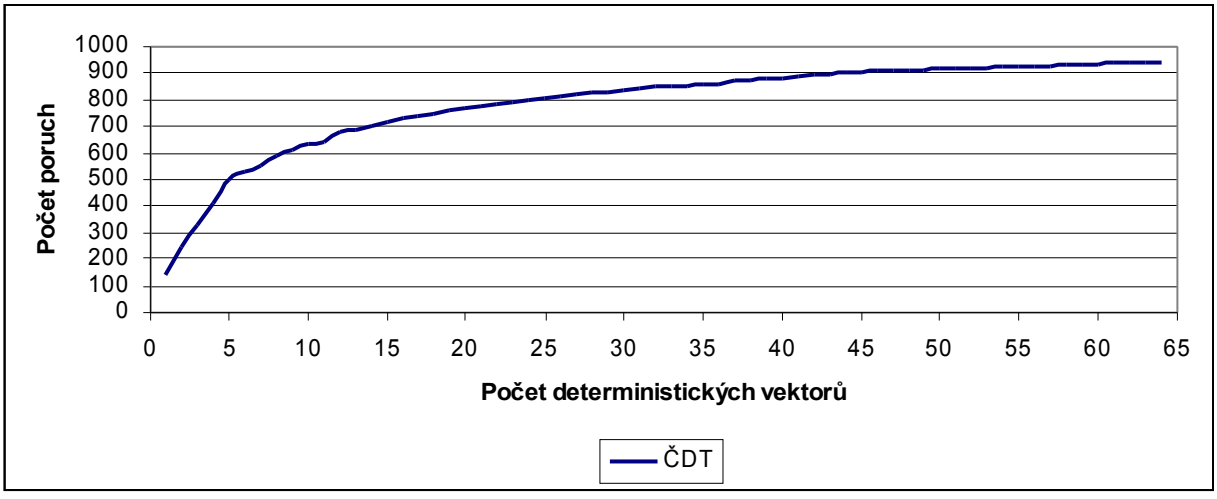
V tabulce 3.2 níže je uveden počet nutných deterministických vektorů pro dané obvody pro detekování všech detekovatelných poruch a grafy 3.1 až 3.6 zobrazují nárůst počtu těchto poruch.

bench	počet deterministických vektorů
c499	53
c880	56
c1908	123
c1355	84
c5315	123
s838	100

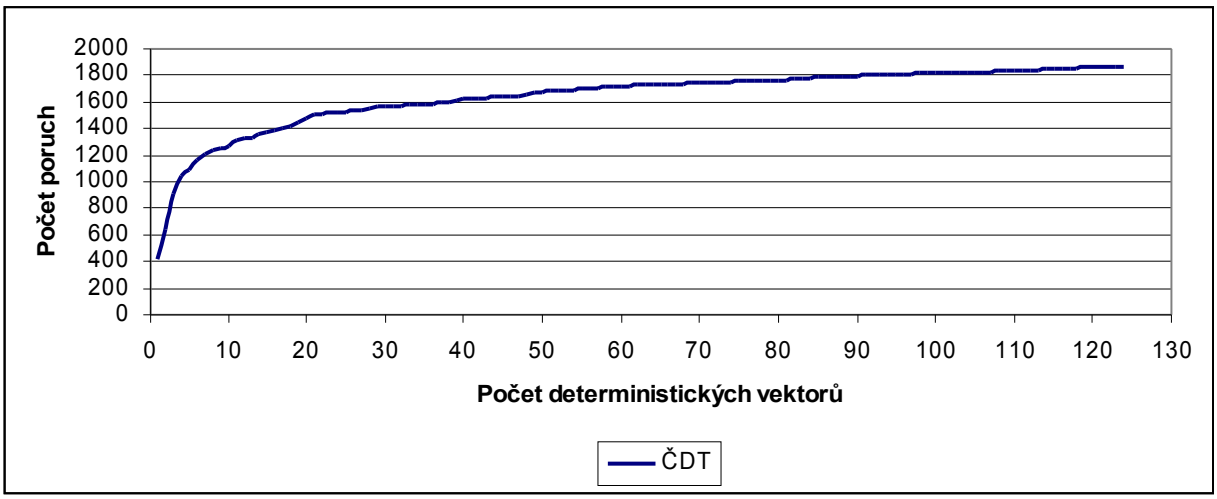
Tab. 3.2 Počty deterministických vektorů



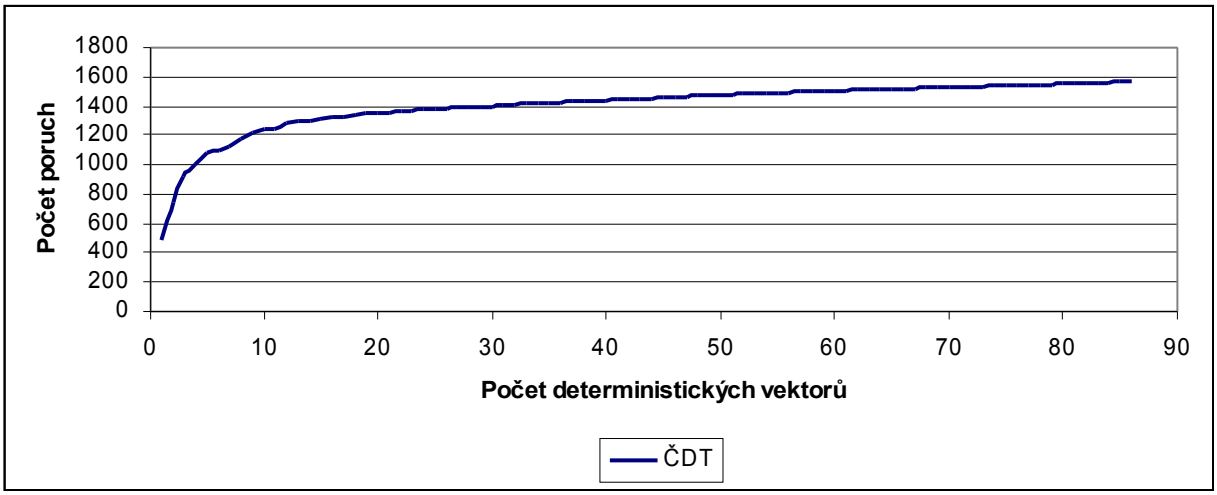
Graf 3.1 Vývoj počtu detekovaných poruch při ČDT (obvod c499)



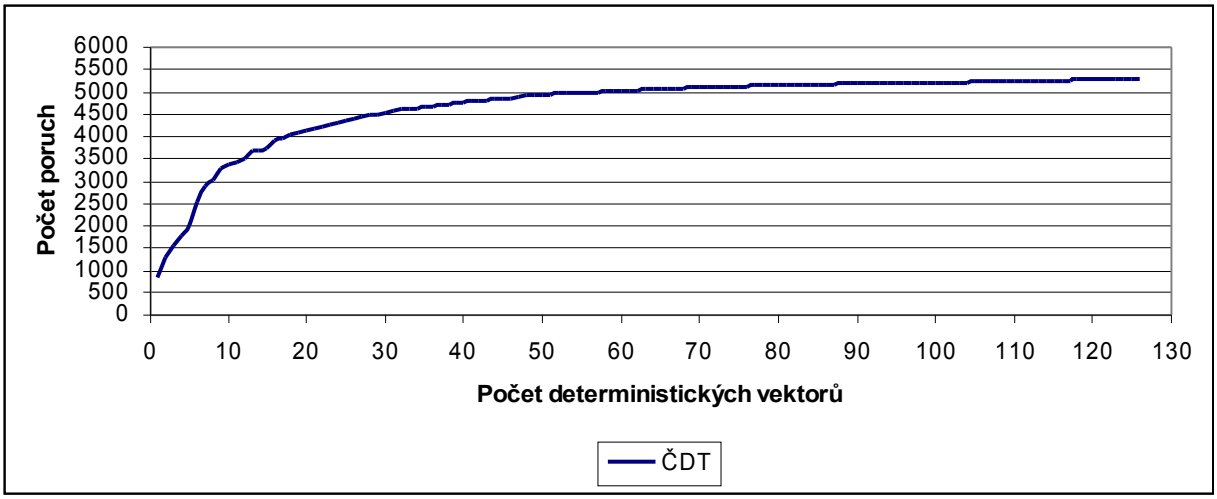
Graf 3.2 Vývoj počtu detekovaných poruch při ČDT (obvod c880)



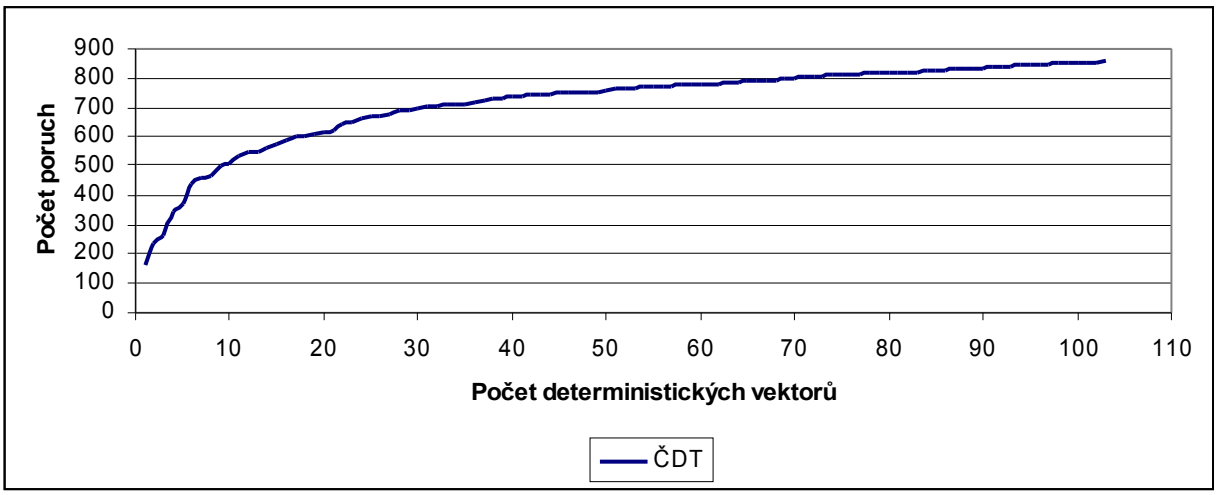
Graf 3.3 Vývoj počtu detekovaných poruch při ČDT (obvod c1908)



Graf 3.4 Vývoj počtu detekovaných poruch při ČDT (obvod c1355)



Graf 3.5 Vývoj počtu detekovaných poruch při ČD (obvod c5315)



Graf 3.6 Vývoj počtu detekovaných poruch při ČD (obvod s838)

3.5 Čistě pseudonáhodné testování (ČPT)

Z pohledu této práce se jedná o úplně základní a nejjednodušší algoritmus, nicméně jeho dílčí funkcionalita, zejména realizace následujícího stavu (vznik dalšího testovacího vektoru) krokem LZPR, je použita i ve vybraných dalších algoritmech. Konkrétně při všech variantách reseedingu.

Jak již bylo zmíněno v úvodu, jedná se o základní typ generátoru, pracující s generujícím polynomem, viz. 3.3.1.1, seedem, což je počáteční nastavení "D" klopných obvodů, a s prvky XOR, umístěných v obvodu právě dle generujícího polynomu. Maximální počet různých generovaných vektorů, řekněme délka sekvence, je roven 2^n-1, kde *n* je počet paralelních výstupů LZPR a počet vstupů obvodu.

Konkrétní délka sekvence závisí na seedu i na polynomu a od určitého okamžiku se začne celá opakovat, nicméně v nejhorším možném případě budeme potřebovat pro detekci všech detekovatelných poruch délku sekvence rovnou jejímu maximu, 2^n.

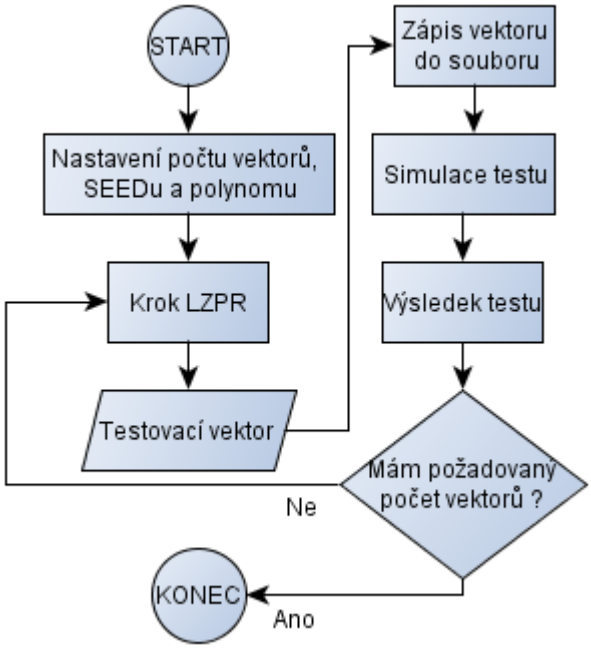
Chceme-li otestovat obvod, který má *n* vstupů, potřebujeme získat testovací vektory také délky *n*, a tedy počet klopných obvodů ve struktuře LZPR bude též rovný *n*. Každým taktem, čili krokem LZPR získáme další nový testovací vektor, který z původního vznikl operací XOR na příslušných bitech (uvažuji opět jen binární vektory). Takto postupujeme proto, abychom zjistili, jak se vyvíjí pokrytí poruch po každém vektoru.

3.5.1 Vlastní implementace algoritmu ČPT

Konkrétní průběh algoritmu se skládá z několika kroků, v nichž se cyklicky opakuje krok LZPR, a končí po vygenerování požadovaného počtu testovacích vektorů. Nad každým testovacím vektorem odsimulujeme test pomocí ATPG Atalanta, blíže v [9] a [10]. Výstupem je několik textových souborů, které zachycují celý průběh algoritmu.

Slovně lze běh algoritmu (po nastavení parametrů) popsat následovně:

1. Použij počáteční seed a udělej krok LZPR, tím získáš testovací vektor
2. Vektor zapiš do testovací sady vektorů a simuluj test
3. Zapiš výsledky testu (počet detekovaných poruch a report z testu) do souborů
4. Proveď další krok LZPR, tím získáš nový testovací vektor
5. Opakuj kroky 2 až 4 pro daný počet kroků; pokud byly detekovány všechny detekovatelné poruchy, skonči



Schema 3.3 Schema algoritmu čistě pseudonáhodného testování

3.5.2 Generování LZPR sekvence pomocí ČPT

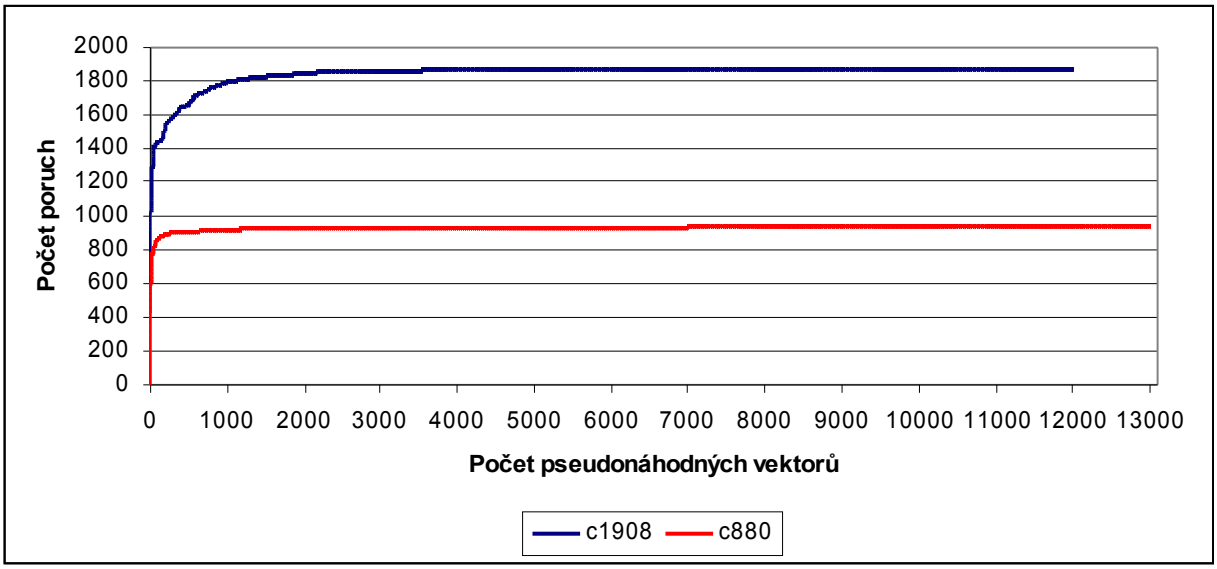
Jelikož je LZPR sekvence použita i v dalších algoritmech napříč celou touto prací a navíc data získaná z měření při použití pokročilých algoritmů často srovnávám právě s daty získanými pomocí ČPT, rozhodl jsem se ověřit správnost dat uvedených v [3].

Nejprve ale pro představu ještě uvádím dvě další měření pro obvody c1908 a c880, jejichž cílem bylo ukázat vývoj počtu detekovaných poruch za použití pseudonáhodného generovaní vektorů, viz. graf 3.7. níže. Jelikož z něj ale není dobře čitelná počáteční fáze testu, kde dochází k razantnímu nárůstu počtu detekovaných poruch, uvádím i graf. 3.8., což jsou ta samá data jako v grafu 3.7., ale zobrazena v logaritmickém měřítku.

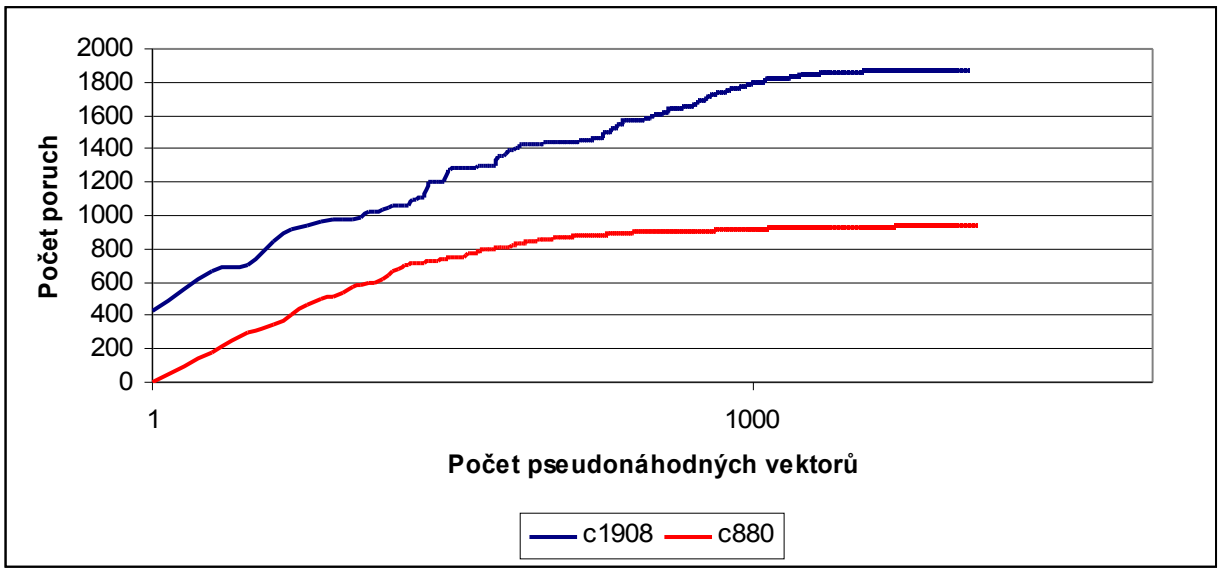
Pro úplnost dodávám, že v těchto dvou měřeních se nepodařilo s nastaveným počtem vektorů obvody plně otestovat. Cílem bylo zobrazit saturační křivku. Počet vektorů, pro něž byl test spuštěn byl roven parametru *avg ověřovaných* dat v tabulce 3.4.

Při ověřování dat, uváděných v literatuře, mi byla vodítkem tabulka 3.3 uvedená níže. Z té jsem si vybral několik obvodů a pro ně provedl měření, která jsem ukončil vždy, když byl obvod plně otestován. Tím rozumím to, že vzniklá sada testovacích vektorů byla schopna detekovat všechny detekovatelné poruchy. Mluvím pak tedy o minimálním neboli nutném počtu vektorů.

V použité literatuře [3] byly výsledky naměřeny pro tisíc různých polynomů a tisíc různých seedů. Já jsem se omezil pouze na tři sta různých polynomů a tři sta různých seedů, ale domnívám se, že i takto získaná data jsou průkazná a lze tedy předpokládat, že pokud byla data pro ostatní obvody získána stejným způsobem, jako pro obvody mnou ověřované, získal bych také při jejich testování shodná, nebo velmi podobná data. Na základě těchto výsledku si dovolím tvrdit, že se mi podařilo ověřit data uváděná v literatuře.



Graf 3.7 Zobrazení vývoje počtu detekovaných poruch při pseudonáhodném testování (obvody c1908 a c880)



Graf 3.8 Zobrazení vývoje počtu detekovaných poruch při pseudonáhodném testování (obvody c1908 a c880)

bench	*i*	range	avg
c17	5	2 – 33	4
c432	36	250 – 120	600
c499	41	300 – 6 K	1 200
c880	60	2 500 – 57 K	13 K
c1355	41	800 – 12 K	2 800
c1908	33	3 K – 77 K	12 K
c2670	233	2.4 M – 12.5 M	4.4 M
c3540	50	5 K – 174 K	32 K
c5315	178	1 400 – 5 K	2 500
c6288	32	33 – 474	131
c7552	207	> 100 M	
s27	7	2 – 192	29
s208.1	18	1 400 – 26 K	6 K
s298	17	100 – 1000	500
s344	24	60 – 1000	250
s349	24	70 – 1000	250
s382	24	150 – 2000	500
s386	13	1 400 – 15 K	3 600
s400	24	120 – 2000	500
s420.1	34	165 K – 4 M	1.4 M
s444	24	130 – 2000	500
s510	25	300 – 2500	900
s526	24	5 K – 67 K	19 K
s641	54	196 K – 3.2 M	1 M
s713	54	294 K – 3.4 M	1 M
s820	23	10 K - 78 K	27 K
s832	23	9 K – 75 K	27 K
s838	67	> 100 M	
s953	45	15 K – 98 K	46 K
s1196	32	196 K – 3.2 M	1 M
s1238	32	21 K – 489 K	118 K

bench	*i*	range	avg
s1423	91	9 K – 138 K	55 K
s1488	14	2500 – 24 K	6 800
s1494	14	2200 – 23 K	5 K
s5378	214	50 K – 196 K	82 K
s9234.1	247	6 M – 30 M	15 M
s13207.1	700	97 K – 879 K	329 K
s15850.1	611	> 10 M	
s35932	1763	150 – 500	230
s38417	1664	> 10 M	
s38584.1	1464	> 1 G	
b01	7	1 – 1100	350
b02	5	1 – 1000	200
b03	34	30 – 2600	700
b04	77	14 K – 330 K	60 K
b05	35	10 K – 70 K	25 K
b06	11	1 – 1200	330
b07	50	220 K – 10 M	3 M
b08	30	2 K – 60 K	13 K
b09	29	4 K – 42 K	16 K
b10	28	300 – 5 K	1700
b11	38	12 K – 160 K	50 K
b12	126	5 – 44 M	13 M
b13	63	700 – 15 K	5 K
b14	277	> 100 M	
b15	485	> 100 M	
b17	1452	> 100 M	
b18	3307	> 100 M	
b19	6666	> 100 M	
b20	522	> 100 M	
b21	522	> 100 M	
b22	767	> 100 M	

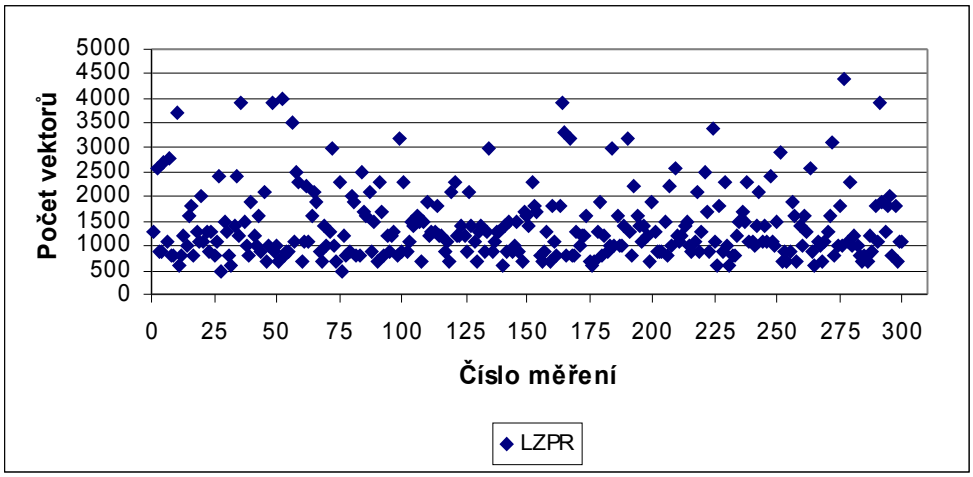
Tab. 3.3 Přehled počtu nutných vektorů pro vybrané obvody (tabulka byla převzata z [3])

Data v tabulce byla získána pouze za použití LZPR sekvence, ne při použití jakéhokoli jiného algoritmu. Význam popisků ve sloupcích je následující: *bench* označuje typ obvodu, *i* udává počet vstupů obvodu, *range* značí rozsah počtu nutných vektorů a konečně *avg* určuje průměrný počet nutných vektorů.

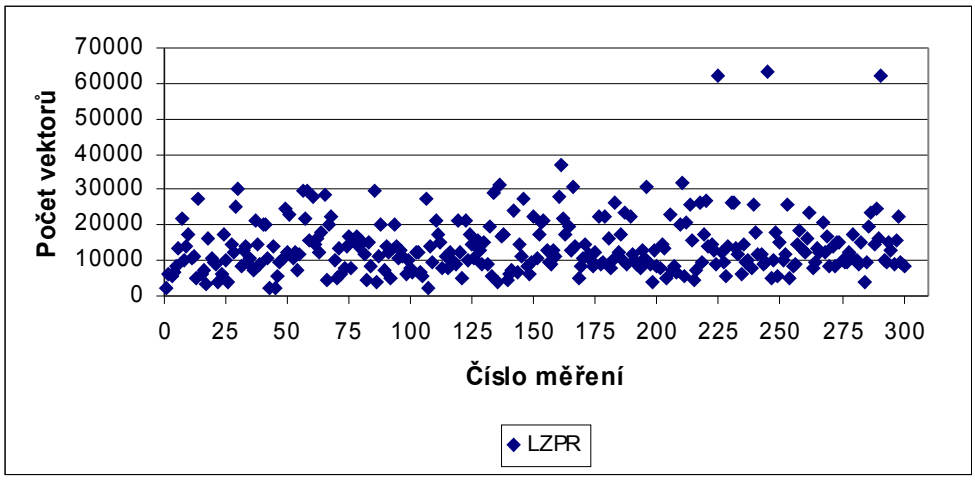
bench	*i*	range	avg	
			ověřovaný	naměřený
c499	41	300 - 6000	1200	1400
c880	60	2500 - 57000	13000	14000
c1908	33	3000 - 77000	12000	11800
c1355	41	800 - 12000	2800	3100
c5315	178	1400 - 5200	2500	2800

Tab. 3.4
Srovnání výsledků uváděných v literatuře s výsledky naměřenými

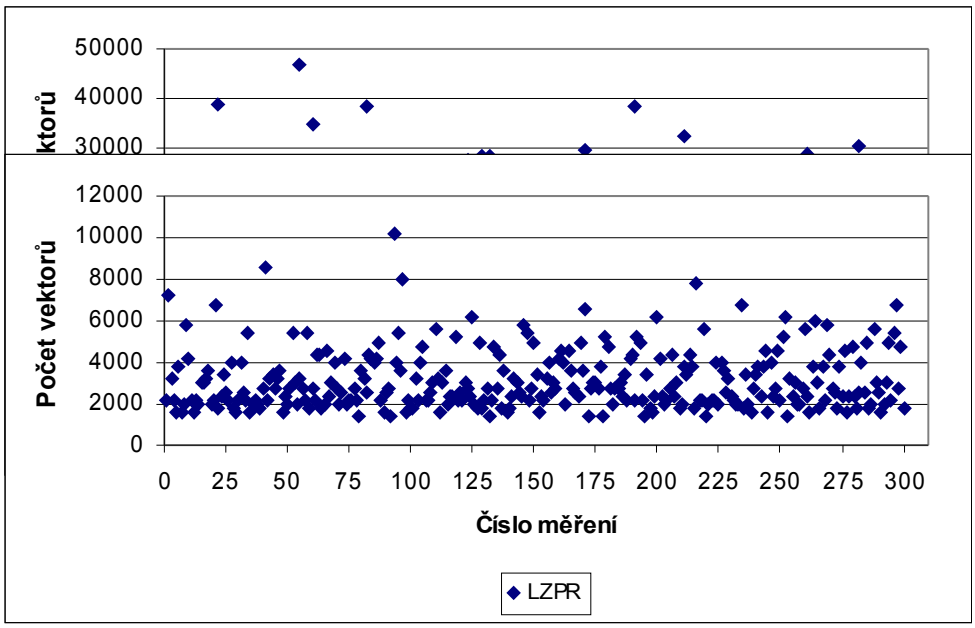
Konkrétní grafy mých měření pro vybrané obvody c499, c880, c1908, c1355 a c5315 jsou zobrazeny níže.



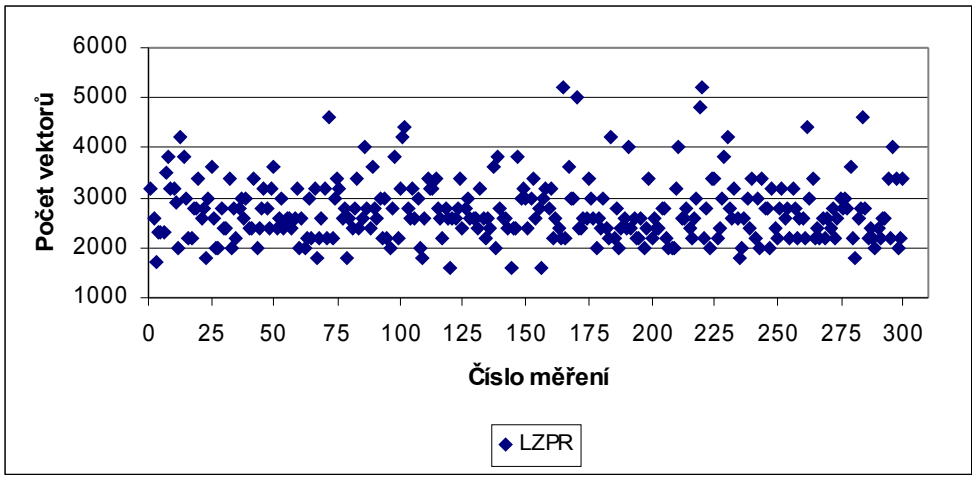
Graf 3.9
Rozptyl hodnot počtu vektorů pro detekci všech detekovat. poruch (obvod c499)



Graf 3.10
Rozptyl hodnot počtu vektorů pro detekci všech detekovat. poruch (obvod c880)



Graf 3.12
Rozptyl hodnot počtu vektorů pro detekci všech detekovat. poruch (obvod c1355)



Graf 3.13
Rozptyl hodnot počtu vektorů pro detekci všech detekovat. poruch (obvod c5315)

4. Reseeding

Z názvu algoritmu vyplývá, že se jedná o změnu seedu, čili o změnu výchozího nastavení obvodu za běhu algoritmu. V úvahu připadají dva případy reseedingu - multipolynomiální [5], [6] a jednopolynomiální [7]. Při multipolynomiálním reseedingu se mění jak seed, tak i polynom (a to i opakovaně), při jednopolynomiálním se mění pouze seed (i opakovaně) a polynom je fixní.

Při vlastní implementaci algoritmu jsem se zaměřil pouze na variantu jednopolynomiálního reseedingu, ovšem v několika modifikovaných verzích. Jednotlivé verze se mezi sebou liší tím, za jakých podmínek má nastat reseeding. Motivací k různým verzím téhož byl fakt, že výsledky měření, uvedené v podkapitole 4.4, nepotvrdily předpokládané chování vývoje detekce poruch, a proto jsem zkoušel i další možné přístupy. Jednotlivé verze implementace jsou popsány níže.

Motivací proč vůbec používat reseeding, nebo obecně jakýkoliv jiný algoritmus, odlišný od prostého čistě pseudonáhodného testování je, jak už bylo zmiňováno v úvodu, najít určitý kompromis mezi délkou testu, objemem dat, nárocích na paměť, případně dalších parametrech za

podmínky, že budou detekovány všechny detekovatelné poruchy.

Při reseedingu se tedy snažíme najít kompromis mezi délkou (dobou) testu a množstvím uchovávaných dat a to opět tak, aby nebyla porušena podmínka, že musí být detekovány všechny detekovatelné poruchy.

4.1 Vlastní implementace jednopolynomiálního reseedingu (var. 1.)

Tato varianta reseedingu by ve skutečnosti měla být uvedena jako druhá, protože přístup, testy a výsledky, zde prezentované, vznikly na základě výsledků varianty uvedené v následující podkapitole 4.4**,** ale vzhledem k tomu, že se jedná o zcela základní a nejprimitivnější použití reseedingu, rozhodl jsem nakonec se tuto variantu zařadit jako první. Výsledky uvedené v podkapitole 4.2 ukazují, jak i jednoduchým použitím reseedingu můžeme redukovat nutný počet vektorů generovaných pomocí LZPR.

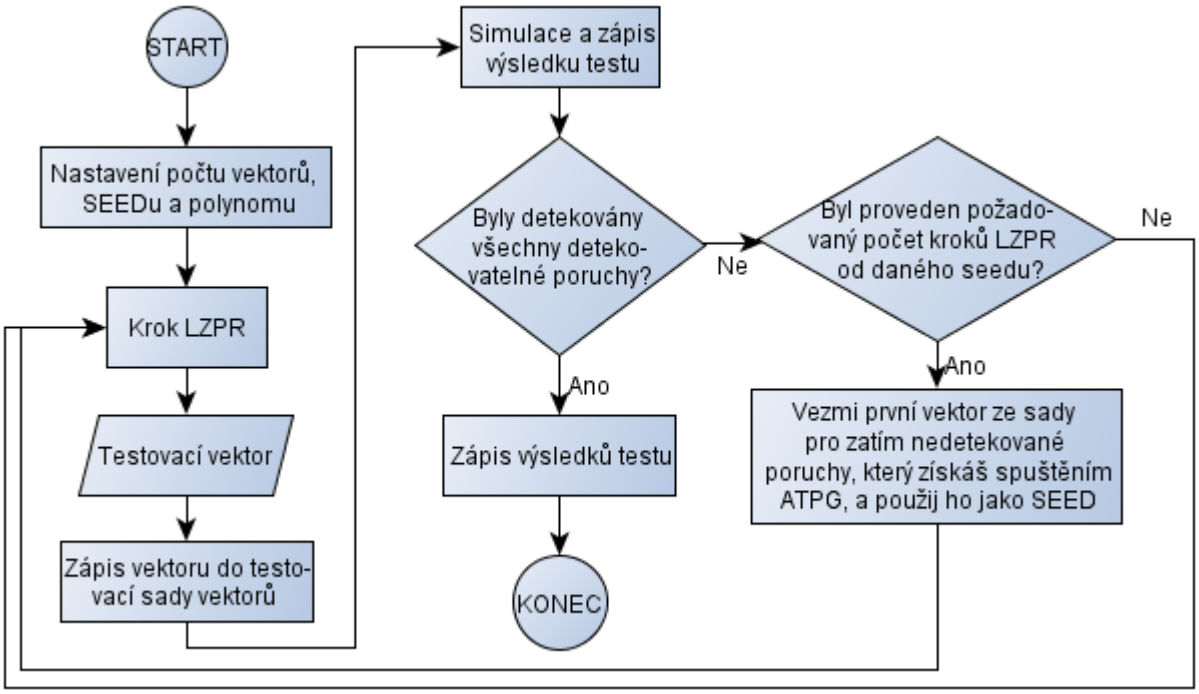
Algoritmus začíná vygenerováním, nebo nastavením seedu a polynomu a nastavením počtu kroků LZPR, čili počtu vektorů, které chci od každého seedu generovat. Vlastní běh algoritmu se opět skládá z dílčích kroků, z nichž některé se cyklicky opakují, a končí při detekování všech detekovatelných poruch.

Slovně lze běh algoritmu (po nastavení parametrů) popsat následovně:

1. Použij aktuální seed a udělej krok LZPR, tím získáš testovací vektor
2. Vektor zapiš do testovací sady vektorů a simuluj test
3. Zapiš výsledky testu (počet detekovaných poruch a report z testu) do souborů
4. Proveď další krok LZPR, tím získáš nový testovací vektor
5. Opakuj kroky 2 až 4 pro daný počet kroků; je třeba zdůraznit, že počet kroků je po celou dobu běhu algoritmu fixní
   Pokud ale byly detekovány všechny detekovatelné poruchy, skonči
6. Pokud nebyly detekovány všechny detekovatelné poruchy, vezmi první vektor pro doposud nedetekované poruchy, který získám spuštěním ATPG, a použij ho jako aktuální seed
7. Pokračuj krokem 1

Schema algoritmu je uvedeno na následující stránce.

Schema 4.1 Schema algoritmu jednopolynomiálního reseedingu (var. 1.)



Zásadní rozdíl oproti algoritmu ČPT je ten, že algoritmus zde končí ne po vygenerování požadovaného počtu vektorů, ale ve chvíli, kdy se podaří detekovat všechny detekovatelné poruchy, což při použití ČPT může být pro složitější obvody "nemožné".Připomenu, že pro s838, je potřeba pro detekci všech detekovatelných poruch přes sto milionů PR vektorů.

Použitím algoritmu uvedeného výše razantně (ve srovnání s algoritmem ČPT) snížíme počet testovacích vektorů, paměťovou i časovou náročnost, nicméně pořád zde chybí rozumný systém vyhodnocování podmínky kdy má reseeding nastat. Je tak potlačen požadavek hledání kompromisu mezi nutným počtem změn v obvodu a počtem použitých testovacích vektorů.

Porovnání výsledků výše zmíněného algoritmu s výsledky algoritmu ČPT jsou uvedeny níže.

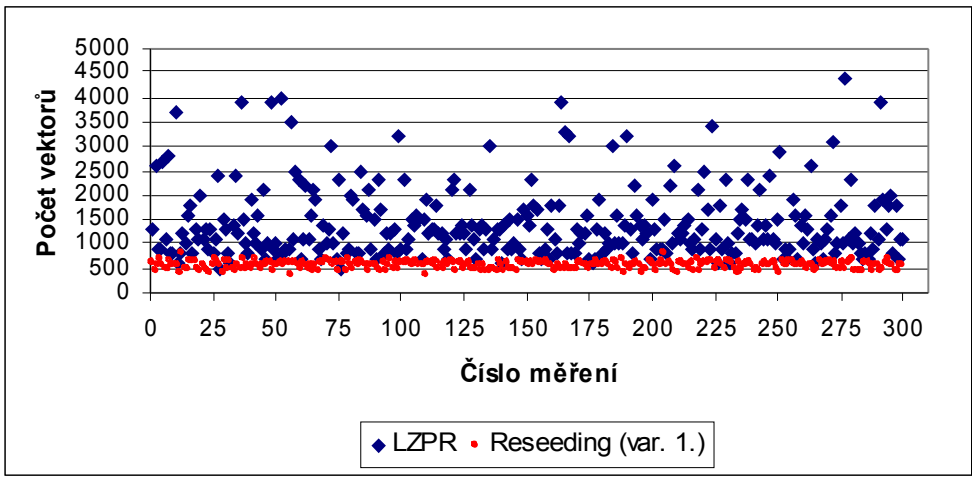
4.2 Porovnání výsledků ČPT s výsledky jednopolynomiálního reseedingu (var. 1.)

V této podkapitole se zaměřuji na porovnání počtu použitých vektorů pro detekci všech detekovatelných poruch při použití ČPT a při použití první varianty jednopolynomiálního reseedingu.

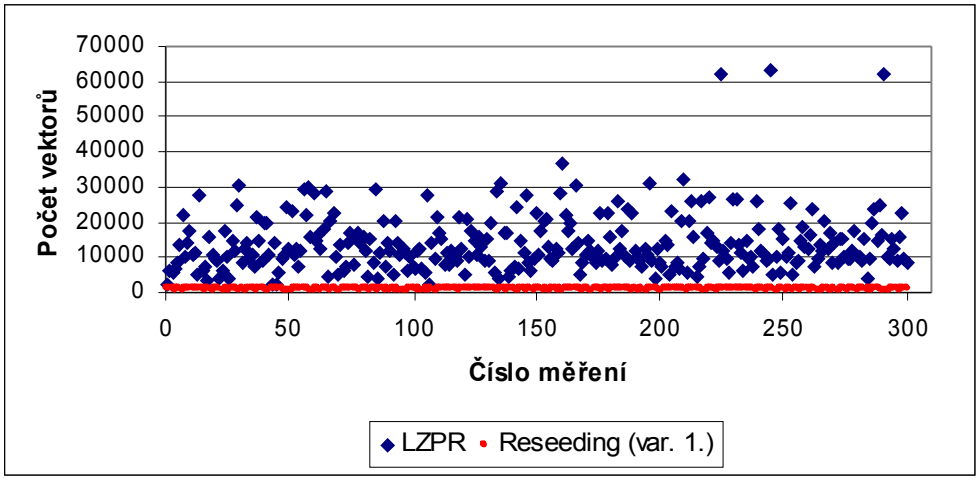
Z výsledků měření je zřejmé, zvláště pak pro složitější obvody, že použitím i nejjednodušší varianty reseedingu lze dosáhnout razantního snížení počtu nutných pseudonáhodných vektorů. Chybí zde ovšem rozumný mechanismus určování zda byl, či nebyl reseeding zbytečný. Zbytečný v tom smyslu, že pokud bychom dále prováděli jen kroky LZPR, mohli jsme detekovat stejný, nebo i větší počet poruch, než kolik jich bylo detekováno v rámci běhu daného reseedingu. Je tedy klidně možné, že ačkoli jsem použil (např. pro obvod c1908) reseedingů desítky, mohl jsem si při vhodně zvolených parametrech vystačit s počtem reseedingů řádově v jednotkách.

Měření, kde uplatňuji složitější vyhodnocování a uvažuji různé hodnoty nastavení parametrů jsou uváděna dále.

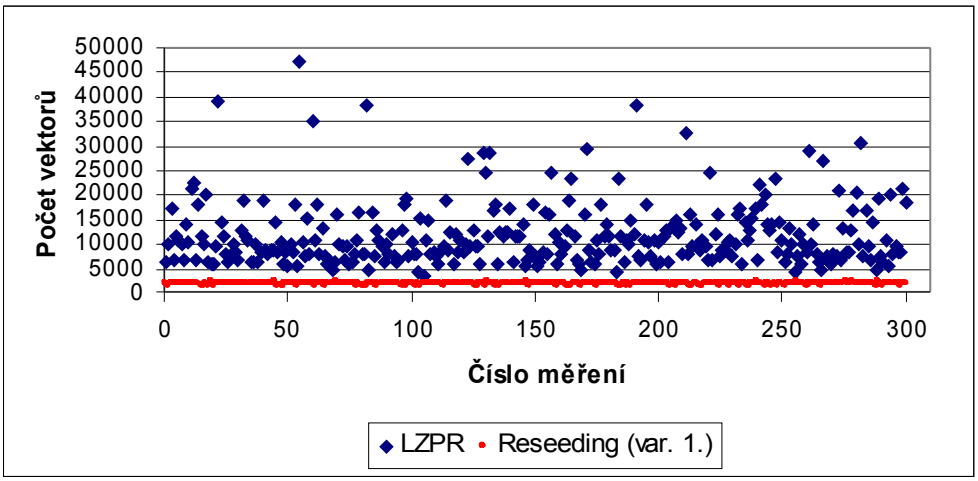
V tomto případě bylo opět provedeno 300 testů pro 300 různých polynomů a to pouze pro reseeding. Uvažovány byly ty samé obvody jako v podkapitole 3.6.2 a stejně tak i data pro ČPT byla převzata z téže podkapitoly. Srovnání počtu vektorů je uvedeno v Tab. 4.1 níže.



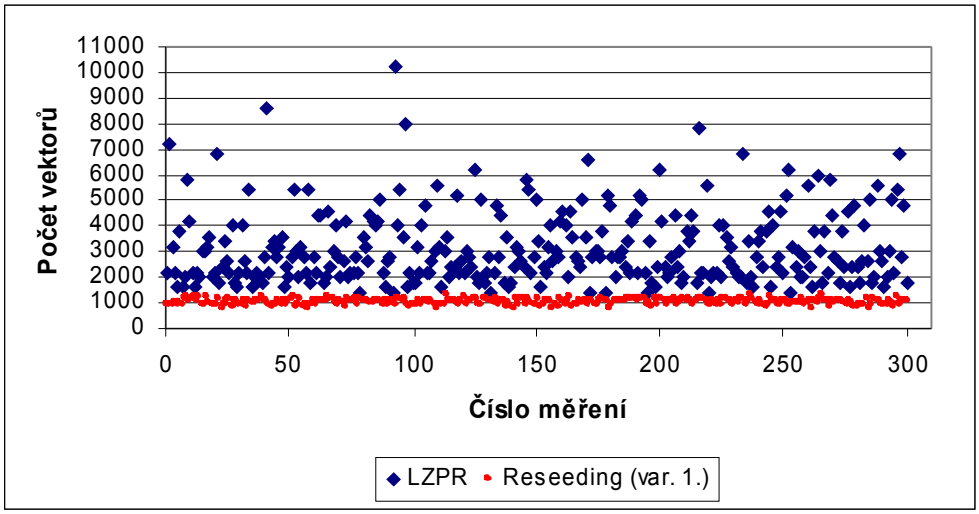
Graf 4.1
Porovnání počtu vektorů pro detekci všech detekovat. poruch (obvod c499)



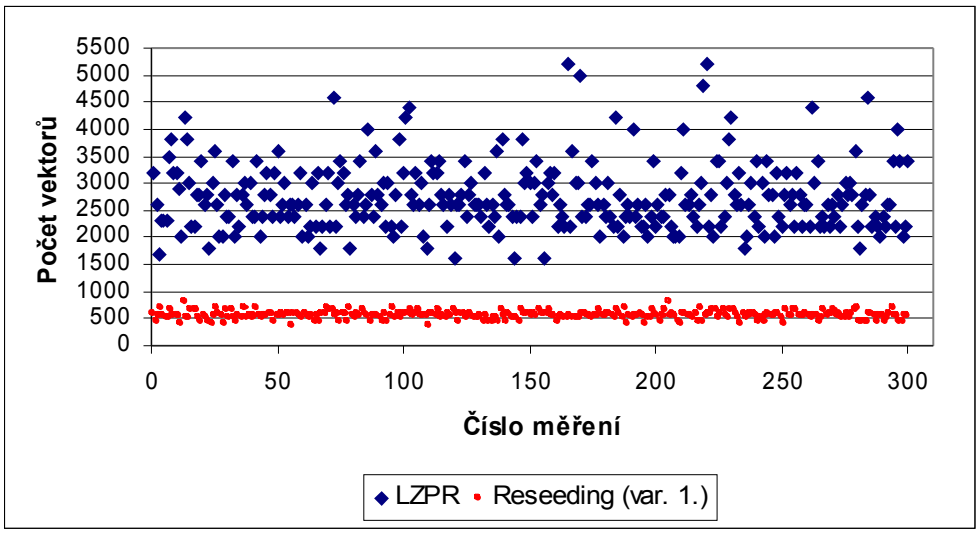
Graf 4.2
Porovnání počtu vektorů pro detekci všech detekovat. poruch (obvod c880)



Graf 4.3
Porovnání počtu vektorů pro detekci všech detekovat. poruch (obvod c1908)



Graf 4.4
Porovnání počtu vektorů pro detekci všech detekovat. poruch (obvod c1355)



Graf 4.5
Porovnání počtu vektorů pro detekci všech detekovat. poruch (obvod c5315)

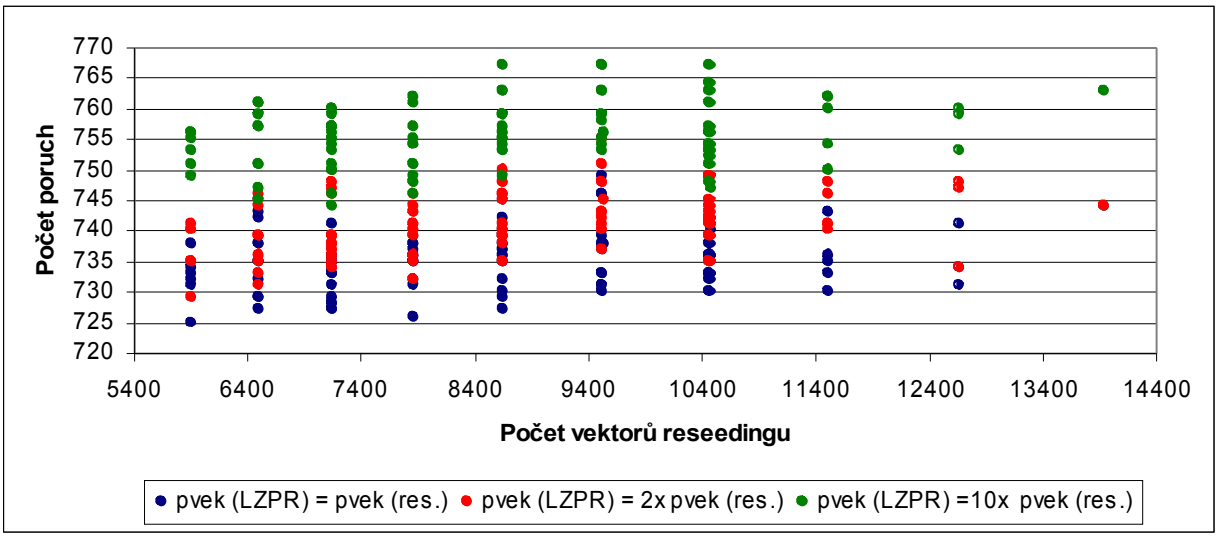
	reseeding (var. 1.)			ČPT			ČDT
bench	min. počet seedů	max. počet seedů	průměrný počet seedů (zaok.)	min. počet PR vektorů (zaok.)	max. počet PR vektorů (zaok.)	průměrný počet PR vektorů (zaok.)	počet deterministických vektorů
c499	7	18	12	300	6000	1300	53
c880	11	40	17	2500	57000	13900	56
c1908	31	51	39	3000	77000	11800	123
c1355	14	26	20	800	12000	3100	84
c5315	2	50	17	1400	5000	2700	123

Tab. 4.1
Srovnání počtu vektorů pro jednotlivé obvody při reseedingu (var. 1.), při ČPT a ČDT

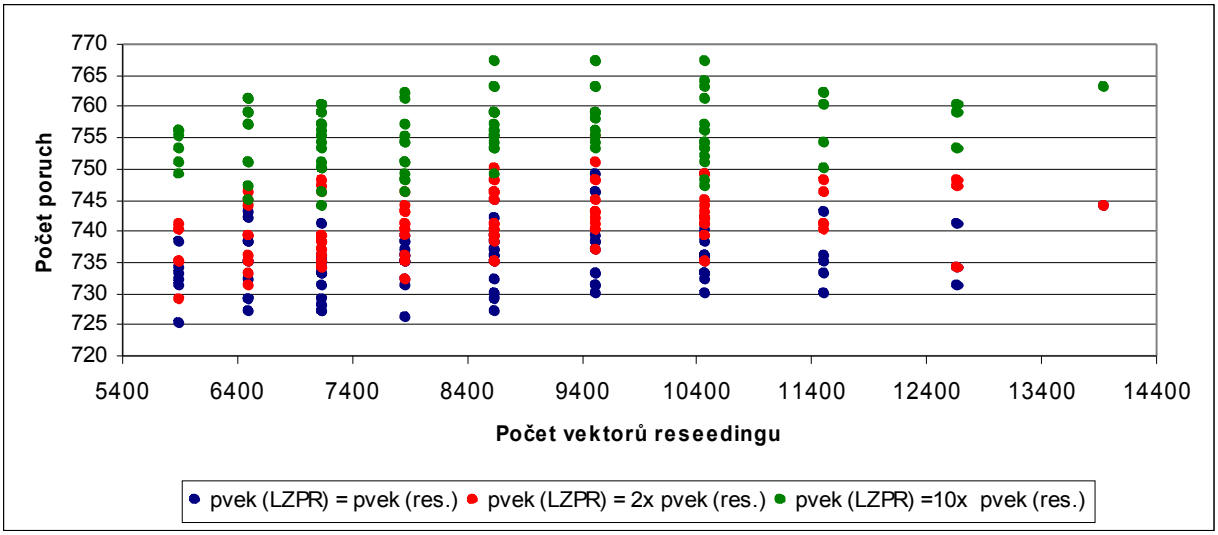
4.3 Porovnání počtu detekovaných poruch při ČPT a reseedingu (var. 1.)

Na základě výsledků prezentovaných v předchozí podkapitole 4.2, jsem se rozhodl provést následující pokus. Řekněme, že jsem při použití reseedingu v kombinaci s PR vektory potřeboval celkově *pvek(res)*=N*M vektorů, kde N nechť je počet seedů a M je počet PR vektorů, které od každého seedu generuji. Zajímalo mne, jaký počet poruch se mi podaří detekovat, pokud vynechám reseeding a budu generovat pouze PR vektory. Provedl jsem tři testy s různými počty PR vektorů, ale pro shodně nastavený seed i polynom ve všech třech případech. Zaprvé *pvek(ČPT)=pvek(res)*, zadruhé *pvek(ČPT)=2*pvek(res)* a konečně *pvek(ČPT)=10*pvek(res)*. Vždy *pvek* značí počet vektorů, *res* označuje použití reseedingu a význam *ČPT* je zřejmý.

Ze sta měření bylo nejvíce potřeba zhruba 13900 vektorů při použití reseedingu a tedy v nejhorším případě 139000 vektorů pro ČPT. Cílem bylo demonstrovat, že ani mnohonásobně vyšší počet PR vektorů bez reseedingu, nedetekuje stejně poruch jako při jeho využití, byť se jedná o jeho nejjednodušší formu. Pro připomenutí uvádím, že obvod s838 má celkem 857 detekovatelných poruch a obvod s713 jich má 543. Grafy níže zobrazují počty detekovaných poruch pro daný počet PR vektorů.



Graf 4.6 Porovnání počtu detekovaných poruch pro dané počty vektorů získané pomocí ČPT s počty vektorů reseedingu (var. 1.) (obvod s838)



Graf 4.7 Porovnání počtu detekovaných poruch pro dané počty vektorů získané pomocí ČPT s počty vektorů reseedingu (var. 1.) (obvod s713)

4.4 Vlastní implementace jednopolynomiálního reseedingu (var. 2.)

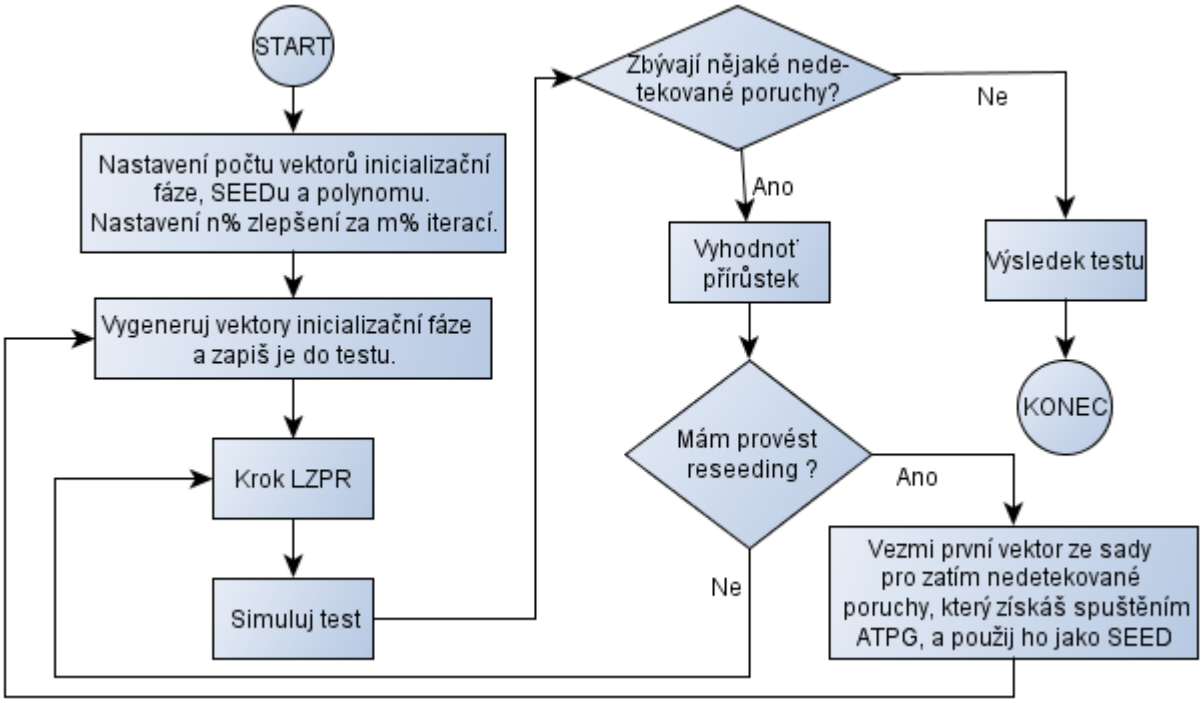
V této variantě algoritmu je pravidlo pro to, kdy má nastat reseeding, dáno procentuálním přírůstkem detekovaných poruch za určité procento iterací. Ptáme se tedy, zda se nám podařilo za posledních *n%* iterací detekovat alespoň o *m%* poruch více, než kolik je detekováno v referenční iteraci. Referenční iteraci určíme tak, že od aktuální iterace odečteme *n%* iterací.

Například: mějme stou iteraci a zajímá nás přírůstek za posledních, řekněme 20% iterací. Znamená to tedy, že počet poruch detekovaných stou iterací srovnáváme oproti počtu poruch detekovaných osmdesátou iteraci.

Vlastní algoritmus začíná vygenerováním, nebo nastavením seedu a polynomu a nastavením počtu kroků LZPR, čili počtu vektorů, které chci od každého seedu generovat před tím, než začnu vyhodnocovat podmínku. Nazvěme tuto fázi před začátkem vyhodnocování fází inicializační. Celý běh algoritmu se skládá z dílčích kroků, z nichž některé se cyklicky opakují, a lze ho slovně popsat (po nastavení parametrů) následovně:

1. Použij aktuální seed a udělej krok LZPR, tím získáš testovací vektor
2. Vektor zapiš do testovací sady vektorů a simuluj test
3. Zapiš výsledky testu (počet detekovaných poruch a report z testu) do souborů
4. Proveď další krok LZPR, tím získáš nový testovací vektor
5. Opakuj kroky 2 až 4 pro daný počet kroků (inicializační fáze), ale pokud byly detekovány všechny detekovatelné poruchy, skonči
6. Pokud nebyly detekovány všechny detekovatelné poruchy, proveď další krok LZPR, odsimuluj test a počet detekovaných poruch v tomto kroku srovnej s počtem detekovaných poruch v referenční iteraci
7. Vyhodnoť podmínku zda byl procentuální přírůstek detekovaných poruch dostatečný a pokud ano, pokračuj krokem 6, pokud ne, přejdi ke kroku 8
8. Proveď reseeding - vezmi první vektor pro doposud nedetekované poruchy, který získám spuštěním ATPG a nový seed použij jako aktuální, pokračuj krokem 1

Algoritmus končí, když jsou detekovány všechny detekovatelné poruchy.



Schema 4.2 Schema algoritmu jednopolynomiálního reseedingu (var. 2.)

Zásadní rozdíl oproti algoritmu uvedenému v podkapitole 4.1 je ten, že zde máme mechanismus, kterým určíme zda a kdy má nastat reseeding, a můžeme tedy hledat kompromis mezi nutným počtem změn v obvodu a počtem použitých testovacích PR vektorů.

Rozdíl oproti algoritmu ČPT je opět ten, že algoritmus zde končí ne po vygenerování požadovaného počtu PR vektorů, ale ve chvíli, kdy se podařilo detekovat všechny detekovatelné poruchy. Tím razantně snížíme počet PR vektorů i časovou náročnost na úkor paměťové.

Výsledky testování výše zmíněného algoritmu jsou uvedeny v podkapitole 4.4.1, ale dovolím si blíže popsat dva grafy a to sice graf 4.10 a graf 4.11 Jedná se o grafy, které vznikly ze sta různých měření (sto různých seedů a polynomů) pro obvody s838 a s713, a které zachycují vývoj a rozptyl počtu detekovaných poruch. Z každého měření jsem vzal minimální a maximální počet aktuálně detekovaných poruch a zobrazil je do grafu. Křivky ohraničují uzavřenou plochu o určité velikosti a je patrné v jakých mezích se budou počty detekovaných poruch nacházet pro jakýkoli polynom a seed pro konkrétní obvody.

4.4.1 Analýza a výsledky jednopolynomiálního reseedingu var. 2.

V této podkapitole se podrobně věnuji analýze a výsledkům dosažených při použití algoritmu jednopolynomiálního reseedingu ve variantě 2. Je to tedy ten případ algoritmu, kdy reseeding nastává v případě, že se nepodařilo za dané procento iterací detekovat o dané procento více poruch. Šlo opět o to najít na základě získaných výsledků optimální parametry nastavení.

Na základě konzultací a výše zmíněné podmínky pro to, kdy má nastat reseeding, jsem měl určitou představu o tom, jak by ideálně měly vypadat výsledky i grafy vývoje detekovaných poruch. V tomto případě jsem tedy hledal nastavení parametrů takové, abych se výsledky získanými co nejvíce přiblížil výsledkům očekávaným. Jak se nakonec ukázalo, a výsledky to potvrzují, výchozí předpoklady byly mylné. I proto byl algoritmus reseedingu modifikován a realizován v další, třetí variantě, která dává nejlepší výsledky.

Očekával jsem tedy, že když už reseeding nastane, vektory vzniklé krokem LZPR od stávajícího aktuálního seedu detekují dostatečné množství dalších poruch. Jinak řečeno, uplatní se i další vektory a ne jen nový seed. Dále jsem očekával, že po každém reseedingu bude zpočátku nárůst počtu detekovaných poruch rychlejší, postupně bude zpomalovat, až se nakonec zpomalí, nebo dokonce ustálí na konkrétní hodnotě natolik, že bude splněna podmínka dalšího reseedingu. Tento cyklus se bude stále opakovat do chvíle, než budou detekovány všechny detekovatelné poruchy s tím, že v každém následujícím reseedingu bude od seedu generovaná sekvence vektorů kratší a celkově bude za danou sekvenci detekováno méně poruch, než za sekvenci předchozí. To je dáno podmínkou kdy má být proveden reseeding. V grafu, zachycujícím vývoj počtu detekovaných poruch, by se tedy, dle původního očekávání, měly objevovat zakulacené "schody".

Skutečný průběh vývoje počtu detekovaných poruch je ale zcela odlišný. Je to dáno tím, že ve fázi ještě před prvním reseedingem je nárůst počtu detekovaných poruch velmi rychlý. Pro všechna měření pro různé obvody byl průběh shodný a to takový, že prvních, nanejvýš dvacet vektorů, detekovalo přes polovinu všech detekovatelných poruch. Na druhou stranu, k detekci posledních deseti procent poruch byl potřeba velký počet seedů. Na každé další dvě, maximálně tři detekované poruchy bylo zapotřebí jednoho seedu. Znamená to tedy, že pouze zbývajících čtyřicet procent poruch mohlo být detekováno dle původních předpokladů, ale ani to se nedělo.

Důvod je zřejmý. Uvažujme situaci, že nastal reseeding a ještě nejsme v poslední fázi, v níž se snažíme dodetekovat posledních deset procent poruch. Po provedeném reseedingu provedeme N kroků LZPR inicializační fáze tak, jak je popsáno v podkapitole 4.4. Nyní začneme měřit a počítáme procentuální přírůstek za posledních *n* procent iterací. Problém ale je, že po reseedingu je nárůst počtu detekovaných poruch pomalý. Pomalý v tom smyslu, že vezmu-li absolutní počet poruch v referenční iteraci a požaduji, aby v aktuální iteraci byl určitý procentuální přírůstek v počtu detekovaných poruch, tak absolutní přírůstek musí být dosti velký. Toto je ale splněno málokdy a pokud ano, tak pouze v několika málo krocích LZPR po fázi inicializační.

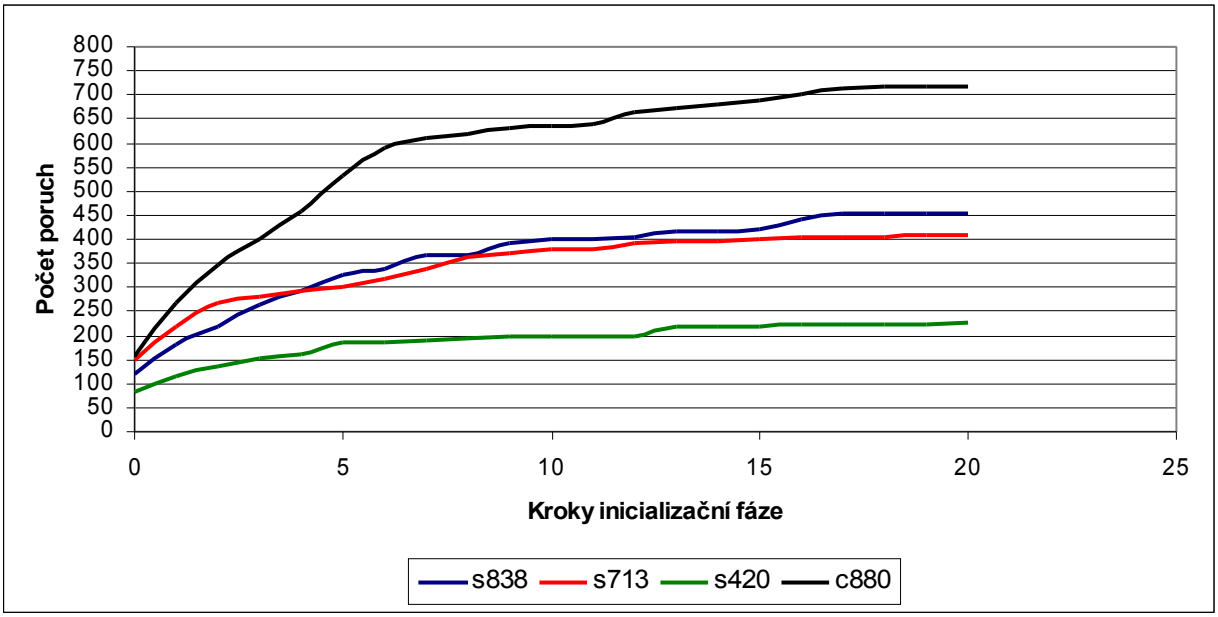
Na druhou stranu, pokud upravím požadavek a snížím hodnotu procentuálního přírůstku v počtu detekovaných poruch, prodloužím tak fázi před prvním reseedingem, detekuji v ní ne padesát, ale třeba sedmdesát procent poruch, fáze posledních deseti procent se nikdy nemění, docílím pouze toho, že první reseeding nastane později, ale běh každého dalšího bude ještě kratší než před úpravou požadavku a ve výsledku bude reseedingů téměř stejně jak před změnou procent. Pokud první fázi zkrátím a detekuji v ní řekněme pouhých třicet procent poruch, má to dopad jen na běh v rámci prvního reseedingu. V něm se totiž podaří dodetekovat tolik poruch, že po jeho skončení bude detekováno zase nejméně padesát procent všech detekovatelných poruch. Výchozí podmínky pro druhý a každý další reseeding jsou tedy zcela shodné s variantou před zkrácením první fáze. Krom toho, první fázi mohu zkrátit buď konkrétním nastavením počtu vektorů, a pak začnu vyhodnocovat podmínku, a nebo nastavím procentuální přírůstek velmi vysoko. Druhá možnost se ale opět projeví i v každém dílčím běhu reseedingu.

Výše zmíněné postřehy a závěry dokládá řada měření níže a ačkoli byla opět provedena celá řada měření, prezentuji výsledky primárně pro obvod s838. Pouze v případech, kdy jsem to považoval za přínosné, uvádím výsledky měření i pro jiné obvody.

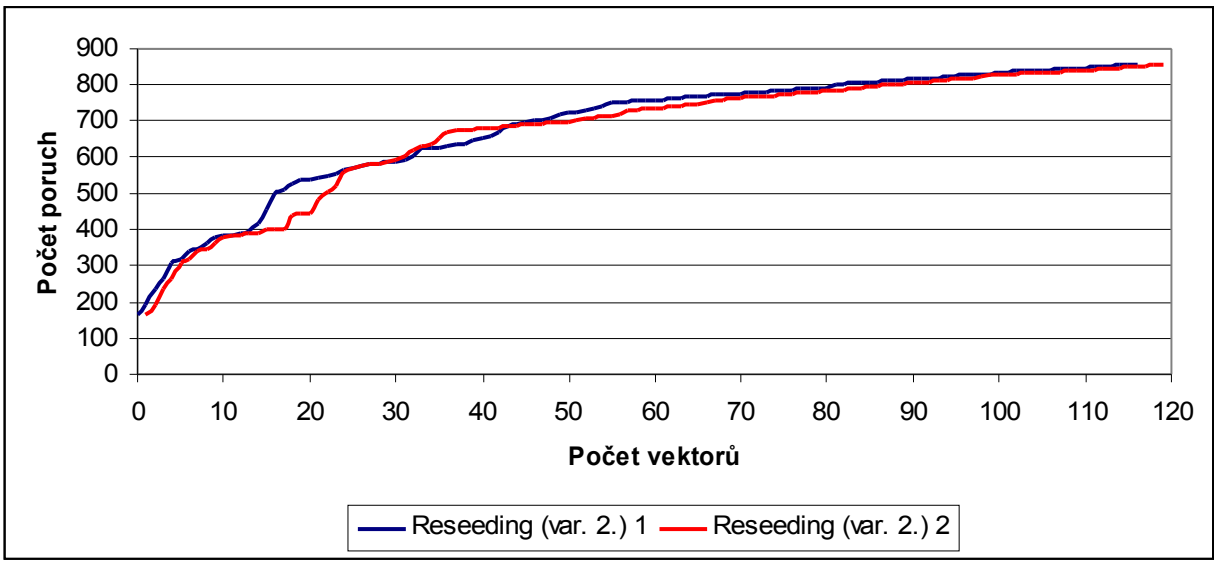
Na závěr lze konstatovat, že tato varianta reseedingu nepřináší kýžené očekávané výsledky.

bench	počet detekovatelných poruch	počet vektorů inicializační fáze (hypoteticky)	detekovaných poruch v inicializační fázi (průměr z měření)
s838	857	20	430
s713	543	20	410
s420	430	20	230
c880	942	20	710

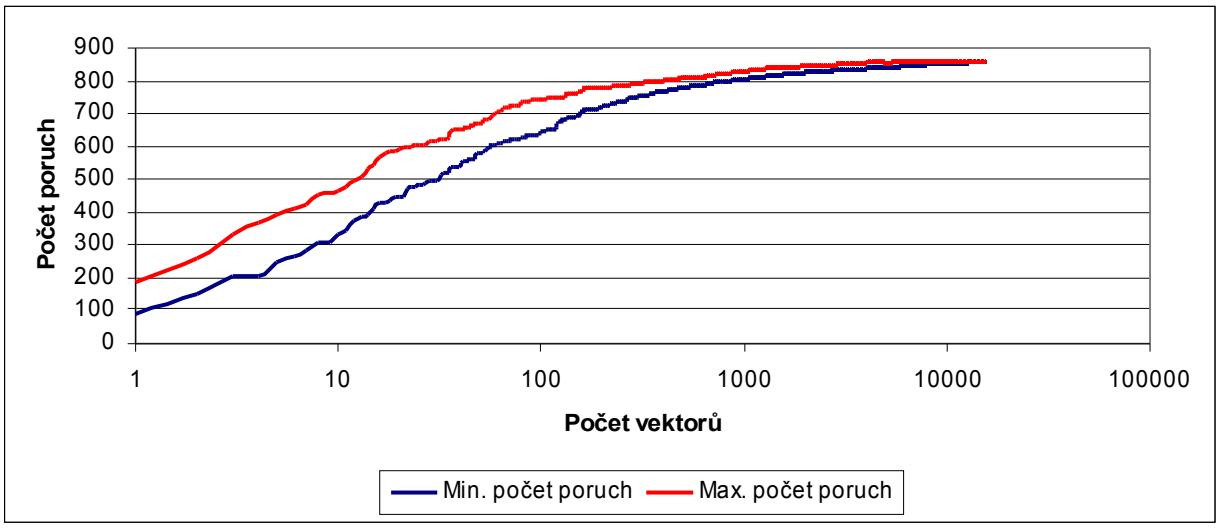
Tab. 4.2 Přehled počtu detekovaných poruch v inicializační fázi pro vybrané obvody



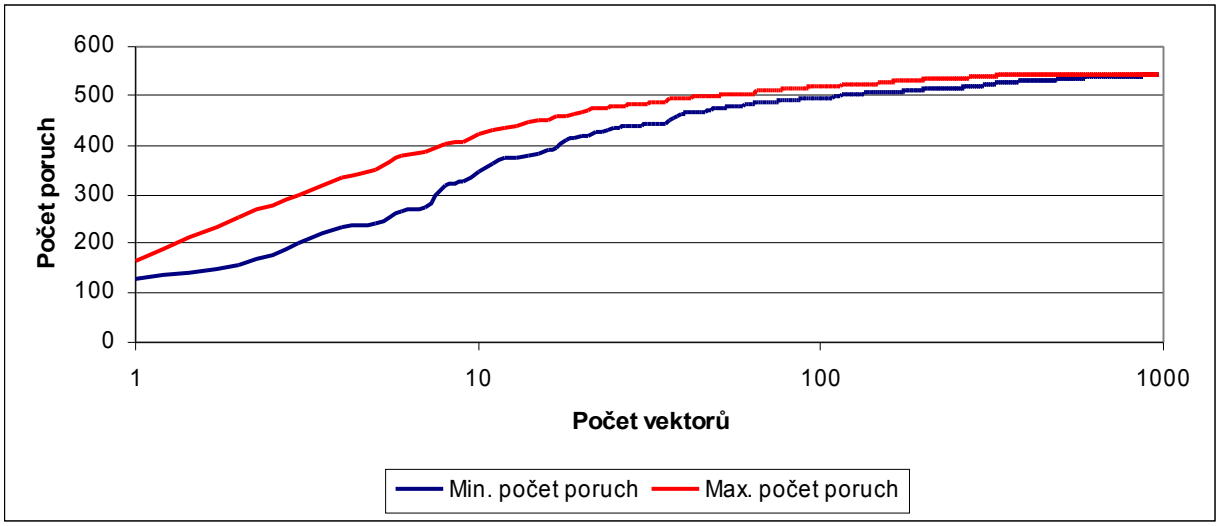
Graf 4.8 Nárůst počtu detekovaných poruch v inicializační fázi pro vybrané obvody



Graf 4.9 Skutečný průběh detekce poruch při reseedingu var. 2. pro dvě různá měření (obvod s838)



Graf 4.10 Rozptyl počtu detekovaných poruch pro 100 měření reseedingu (obvod s838)



Graf 4.11 Rozptyl počtu detekovaných poruch pro 100 měření reseedingu (obvod s713)

4.5 Vlastní implementace jednopolynomiálního reseedingu (var. 3.)

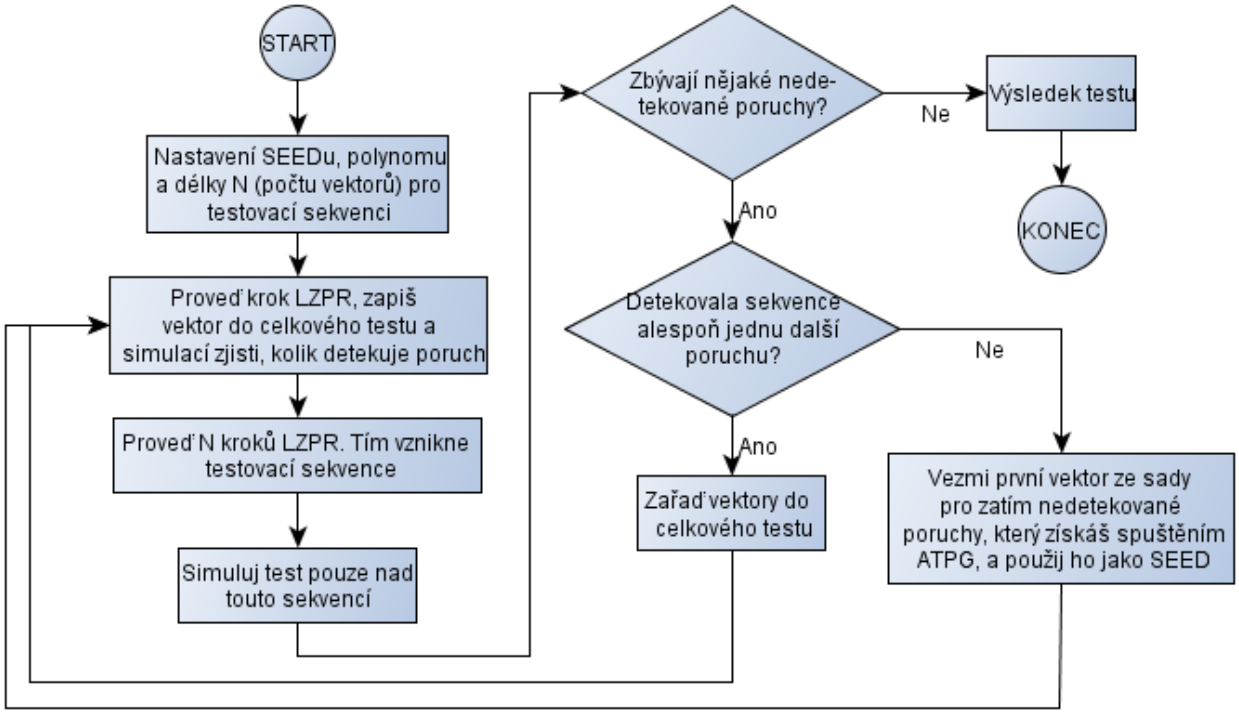
Tento algoritmus přímo vychází z algoritmu uvedeného v předchozí podkapitole, ale oproti předchozí variantě je zde pravidlo pro to, kdy má nastat reseeding, zmírněno. Postačující je když sekvence o N vektorech, vzniklých po sobě jdoucími kroky LZPR, detekuje alespoň jednu další poruchu. Motivací k tomuto testu jsou neuspokojivé výsledky dosažené předchozí variantou algoritmu. Jde o to, že přírůstky detekce poruch, ať už v běhu reseedingu, nebo mezi dvěma po sobě jdoucími reseedingy, byly oproti očekávání mizivé, řádově v jednotkách procent. Tím pádem byla často splněna podmínka pro to, kdy má nastat reseeding, a to se stávalo nadmíru často. Konečně až zde popisovaná, třetí varianta algoritmu reseedingu, dává nejlepší výsledky.

Vlastní algoritmus začíná vygenerováním, nebo nastavením seedu a polynomu a nastavením počtu kroků LZPR, čili počtu vektorů, které chci od každého seedu generovat. Tyto vektory pak zahrnu do sekvence a budu zjišťovat, zda celá tato sekvence detekuje alespoň o jednu poruchu více, než kolik bylo detekováno prvním vektorem reseedingu. Prvním vektorem proto, že tento vektor zcela určitě sám o sobě detekuje alespoň jednu další poruchu. To mám zajištěno tím, že vybírám z vektorů pro zatím nedetekované poruchy, které získám spuštěním ATPG. Pokud tedy celá sekvence detekuje alespoň o jednu poruchu více, zařadím všechny vektory sekvence do celkového testu. Pokud sekvence nedetekuje žádnou další poruchu, vektory by byly nadbytečné a do celkové testovací sady je nezařadím. Algoritmus končí, když jsou detekovány všechny detekovatelné poruchy.

Celý běh algoritmu se opět skládá z dílčích kroků, z nichž některé se cyklicky opakují, a lze ho slovně popsat následovně (po nastavení parametrů):

1. Použij aktuální seed a udělej krok LZPR, tím získáš testovací vektor
2. Vektor zapiš do testovací sady vektorů a simuluj test
3. Zapiš výsledky testu (počet detekovaných poruch a report z testu) do souborů
4. Proveď další krok LZPR, tím získáš nový testovací vektor, ten zařaď do testované sekvence
5. Opakuj krok 4 pro daný počet kroků; pokud ale byly detekovány všechny detekovatelné poruchy, skonči
6. Pokud sekvence detekovala alespoň jednu další poruchu, všechny vektory sekvence přidej mezi vektory celkového testu, poslední vektor sekvence použij jako aktuální seed a pokračuj krokem 1
7. Proveď reseeding - vezmi první vektor pro doposud nedetekované poruchy, který získám spuštěním ATPG a nový seed použij jako aktuální, pokračuj krokem 1

Schema algoritmu je uvedeno na následující stránce.



4.5.1 Analýza a výsledky jednopolynomiálního reseedingu (var. 3.)

Na základě výsledků a analýz předchozích dvou variant reseedingu, jsem implementoval a otestoval reseeding ve variantě třetí. Opět se jedná o změnu seedu na základě vyhodnocení podmínky. Podmínka je taková, že pokud za celou sekvenci PR vektorů délky *N* nebude detekována žádná další porucha, nastane reseeding. Délky sekvence jsem nastavil libovolně od pěti do dvou set, eventuálně až do osmi set PR vektorů pro jednodušší obvody.

Pozn.: Mluvím-li dále v textu o testovacích vektorech, vždy tím myslím PR vektory.

Opět jsem zkoušel i dvě varianty tohoto algoritmu a první z nich byla ta, že jsem celou sekvenci testovacích vektorů rovnou přidal do výsledné sady testovacích vektorů pro daný obvod. To jsem provedl bez ohledu na to, zda sekvence detekovala další poruchu, eventuálně poruchy. Tento způsob jsem ale nakonec zavrhl. Důvod je ten, že se do sady testovacích vektorů dostaly vektory nadbytečné, čili ty, které nedetekovaly žádnou novou poruchu. Pokud ještě navíc byla sekvence dlouhá, nadbytečných vektorů bylo mnoho a nemá smysl uchovávat vektory, které mají nulový přínos. Pouze by se tak zbytečně prodlužovala doba testu.

Ve druhé variantě jsem nejdříve otestoval celou sekvenci vektorů a teprve v případě, že detekovala alespoň jednu další poruchu, přidal jsem ji i k celkové sadě testovacích vektorů. Výhody této varianty jsou zcela zřejmé, a proto ve všech testech používám pouze ji.

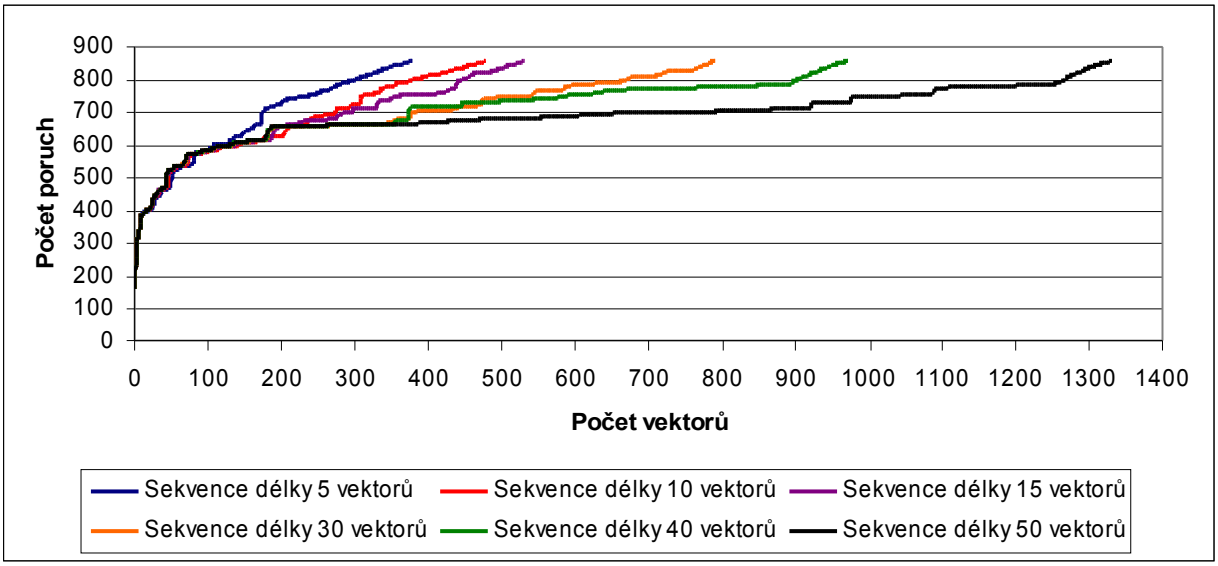
4.5.2 Výsledky měření pro proměnnou délku sekvence PR vektorů

V následujících podkapitolách jsou prezentovány vybrané výsledky měření pro čtyři obvody různé složitosti PR testovatelnosti. Pro každý obvod vždy uvádím dvě měření. V měření se měnily parametry seed a/nebo polynom a délka sekvence. Ke generování sekvence byly použity klasické kroky LZPR. Cílem měření je demonstrovat, jak se mění počet seedů s použitou délkou sekvence.

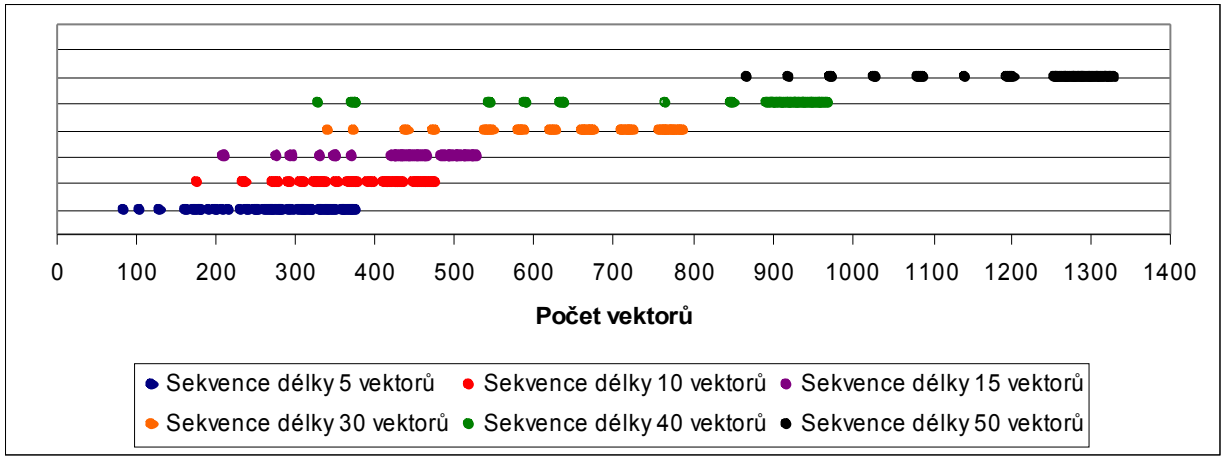
Pro dokreslení ještě zobrazuji, kde v průběhu algoritmu docházelo k reseedingu. Grafy, které toto zachycují, např. graf 4.13, záměrně nemají osu y, která by byla matoucí. Šlo o to, aby místa reseedingu pro různé délky sekvence nebyla zobrazena přes sebe. To jsem vyřešil tak, že jsem tato místa zobrazoval na různých hladinách pomyslné osy y.

4.5.2.1 Výsledky měření pro obvod s838

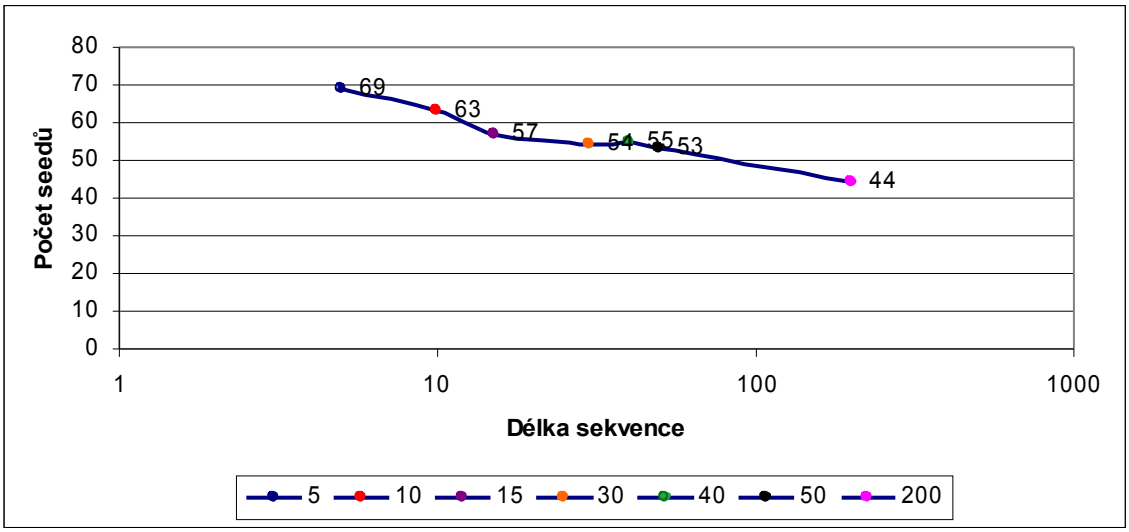
Jedná se o nejhůře PR testovatelný obvod, kterým jsem se v této práci zabýval.



Graf 4.12 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s838)

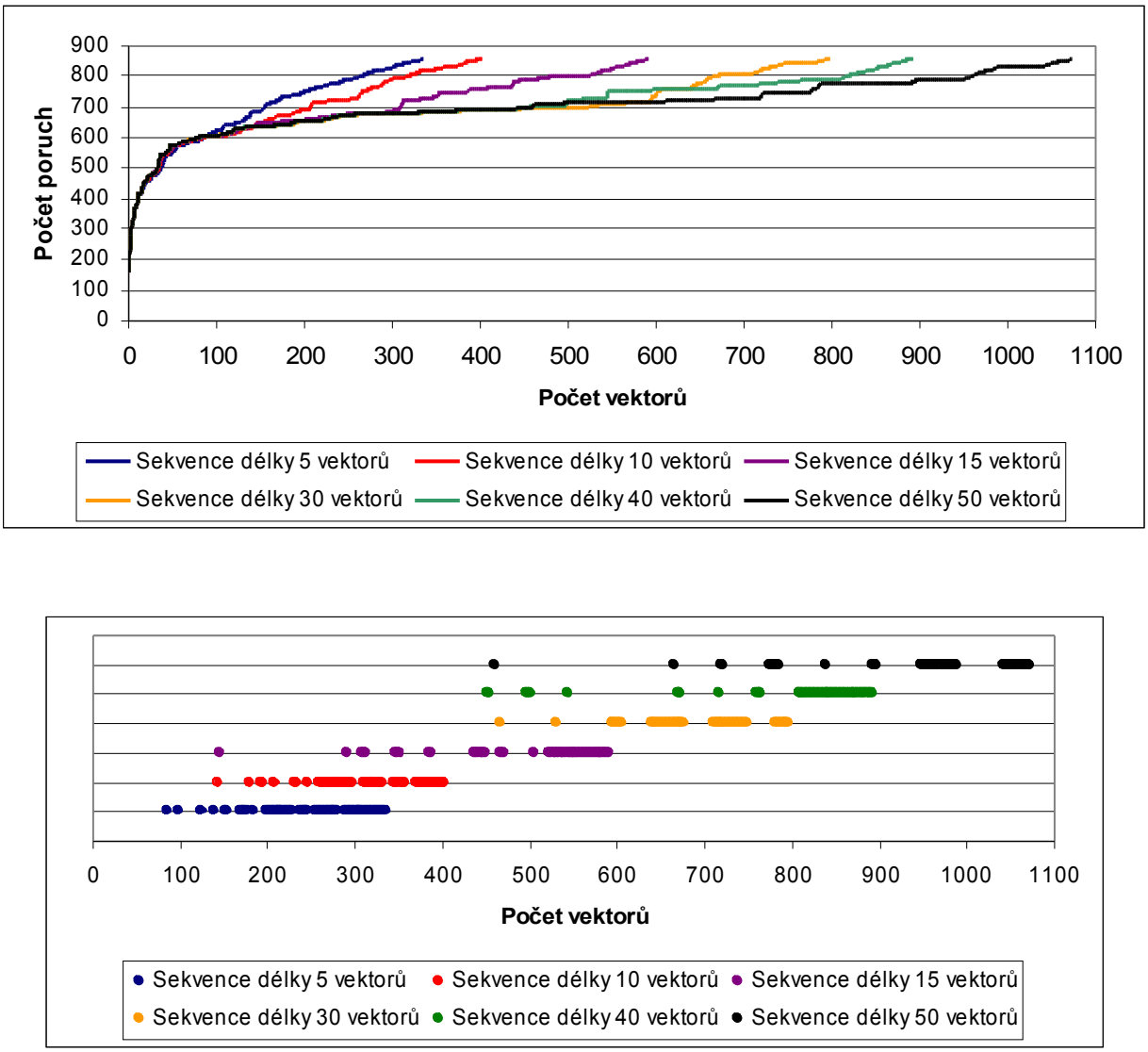


Graf 4.13 Porovnání míst reseedingu dle délky sekvence (obvod s838)

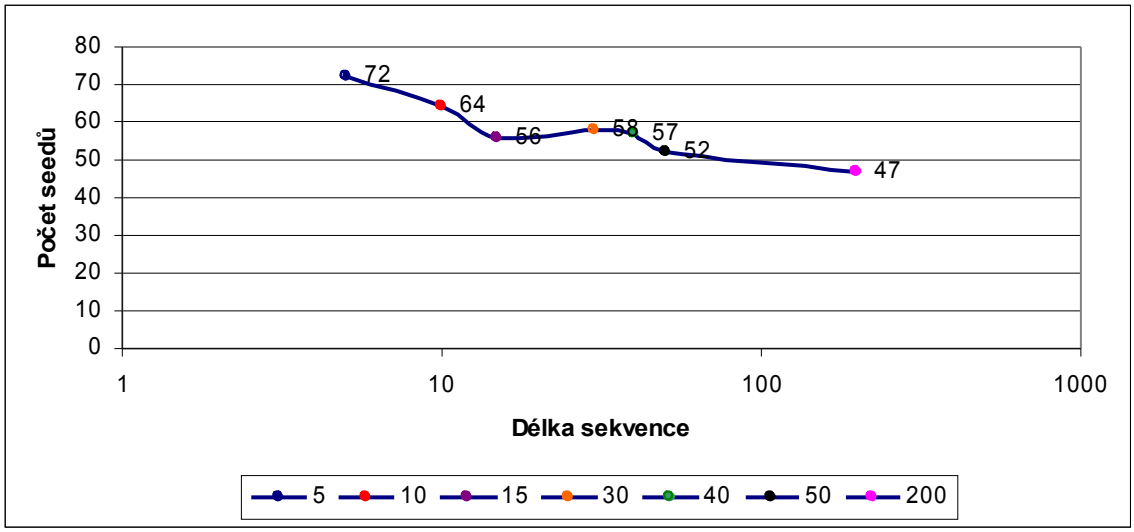


Graf 4.14 Vývoj počtu seedů dle délky sekvence (obvod s838)

Graf 4.15 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s838)



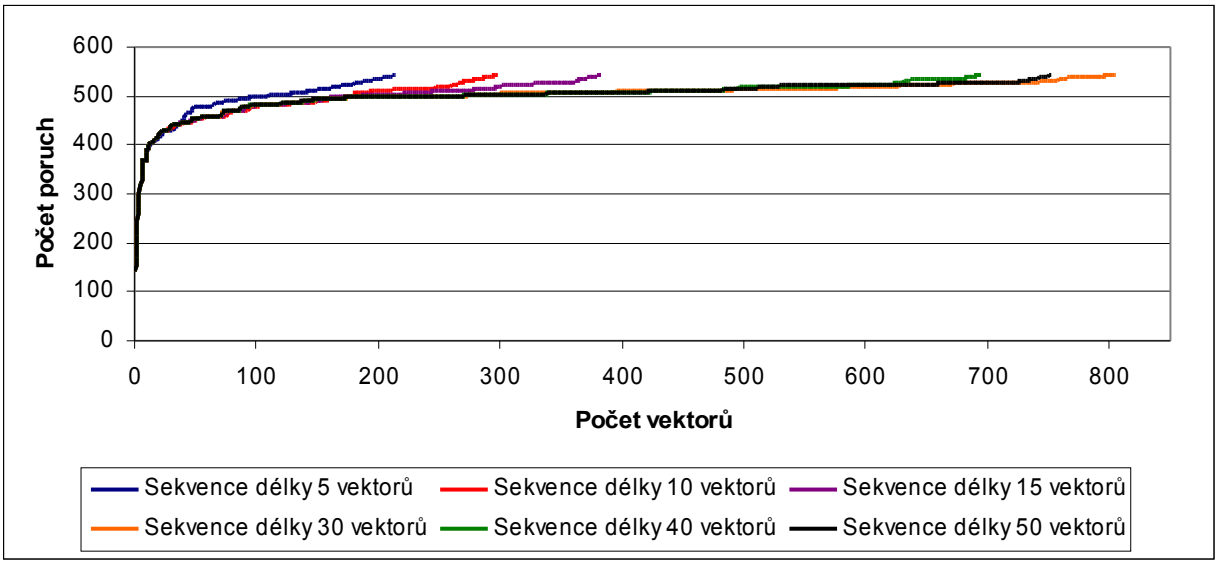
Graf 4.16 Porovnání míst reseedingu dle délky sekvence (obvod s838)



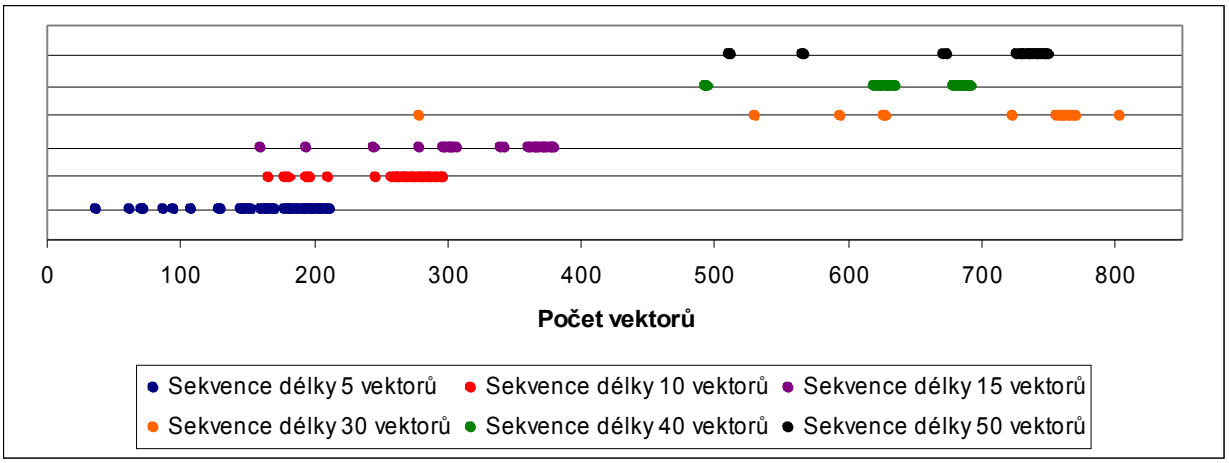
Graf 4.17 Vývoj počtu seedů dle délky sekvence (obvod s838)

4.5.2.2 Výsledky měření pro obvod s713

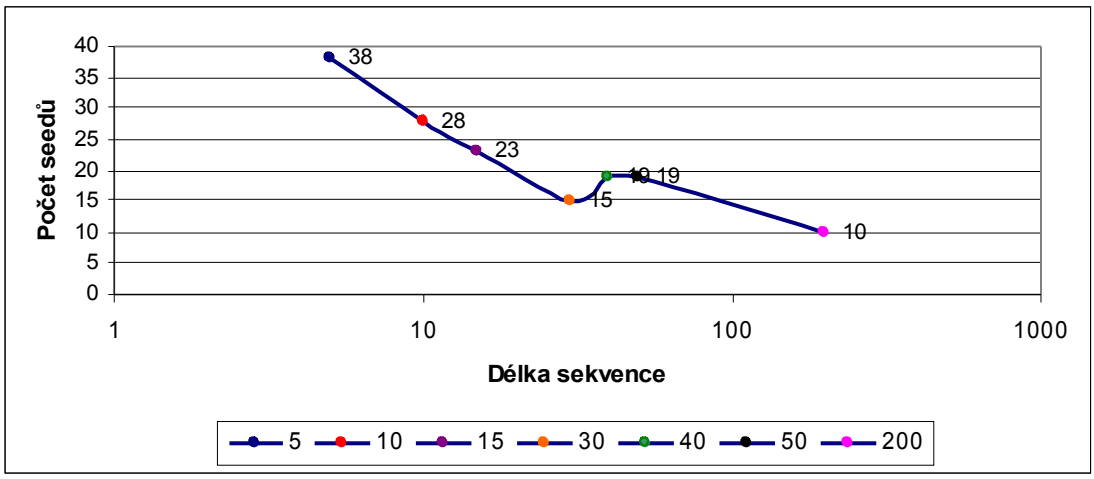
Jedná se o poměrně složitý obvod. Bez reseedingu je potřeba k jeho plnému otestování průměrně jeden milion vektorů.



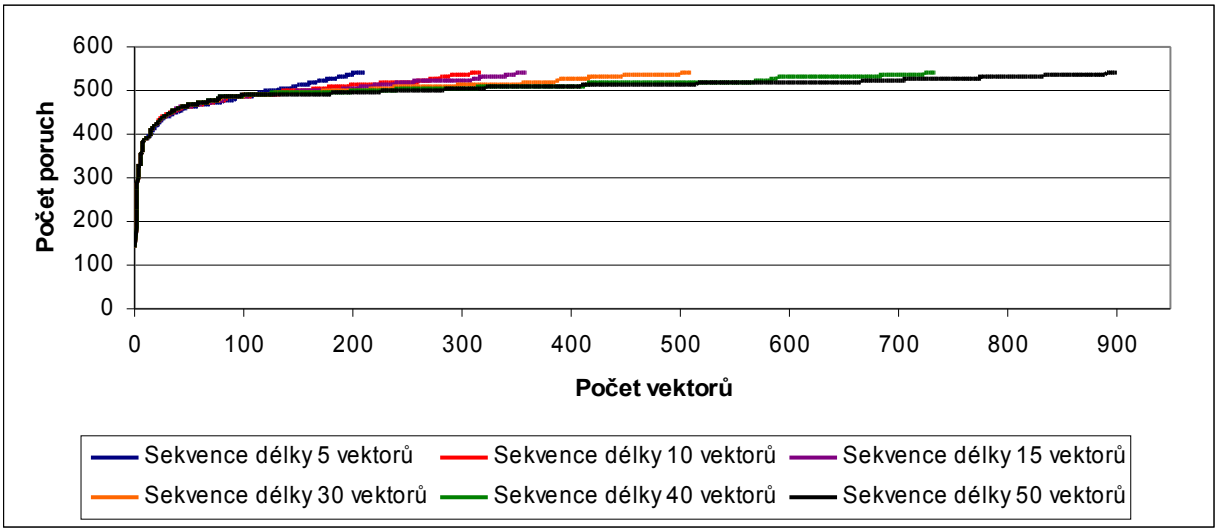
Graf 4.18 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s713)



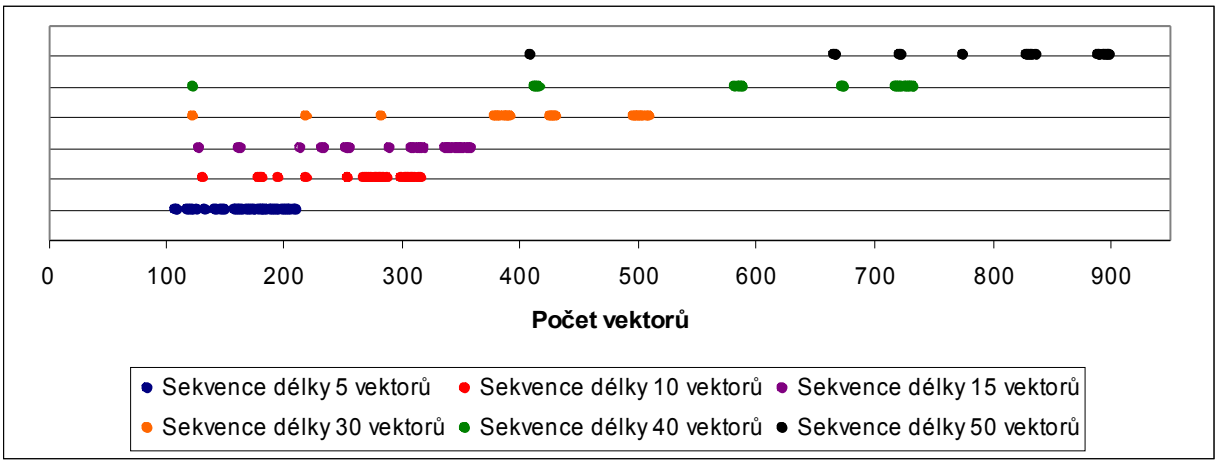
Graf 4.19 Porovnání míst reseedingu dle délky sekvence (obvod s713)



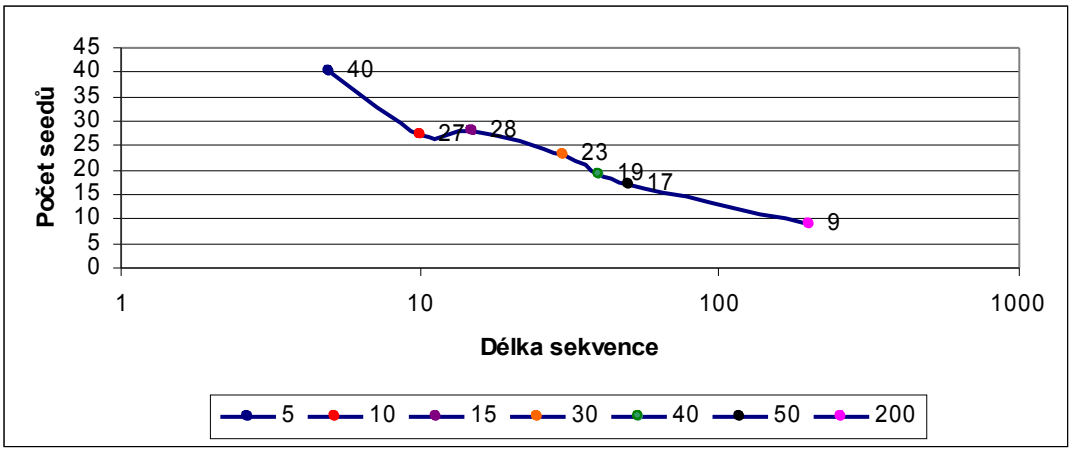
Graf 4.20 Vývoj počtu seedů dle délky sekvence (obvod s713)



Graf 4.21 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s713)



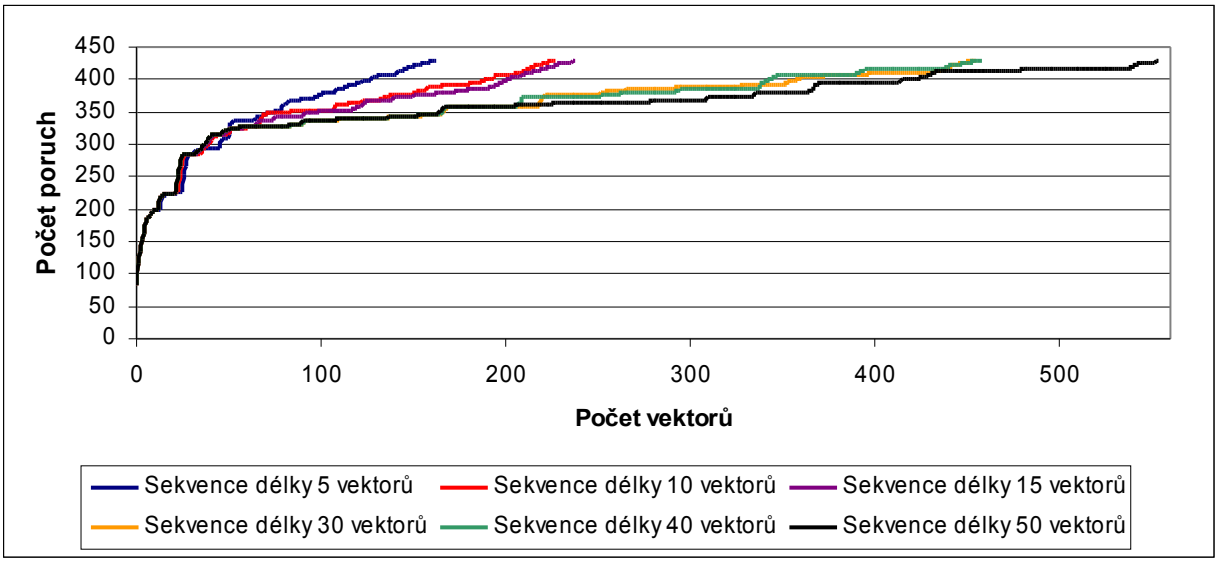
Graf 4.22 Porovnání míst reseedingu dle délky sekvence (obvod s713)



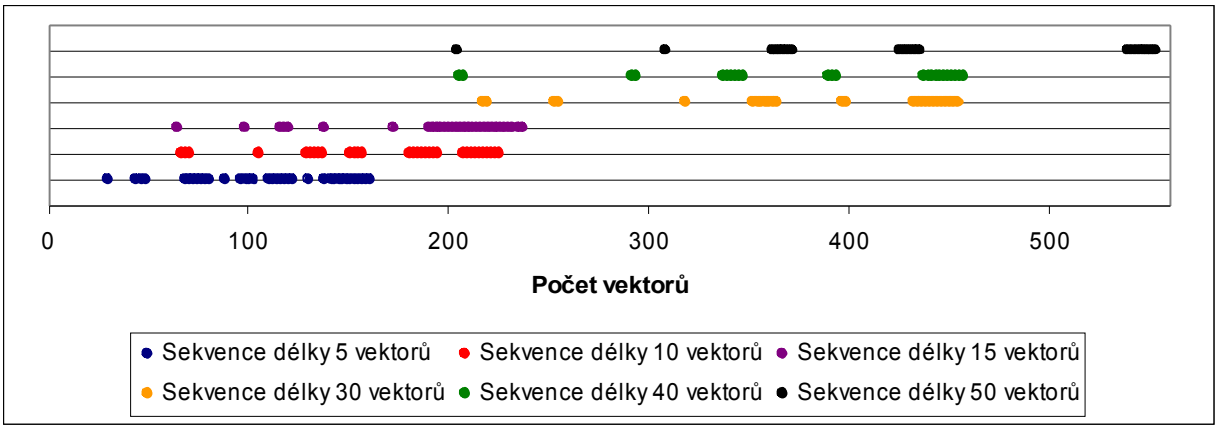
Graf 4.23 Vývoj počtu seedů dle délky sekvence (obvod s713)

4.5.2.3 Výsledky měření pro obvod s420

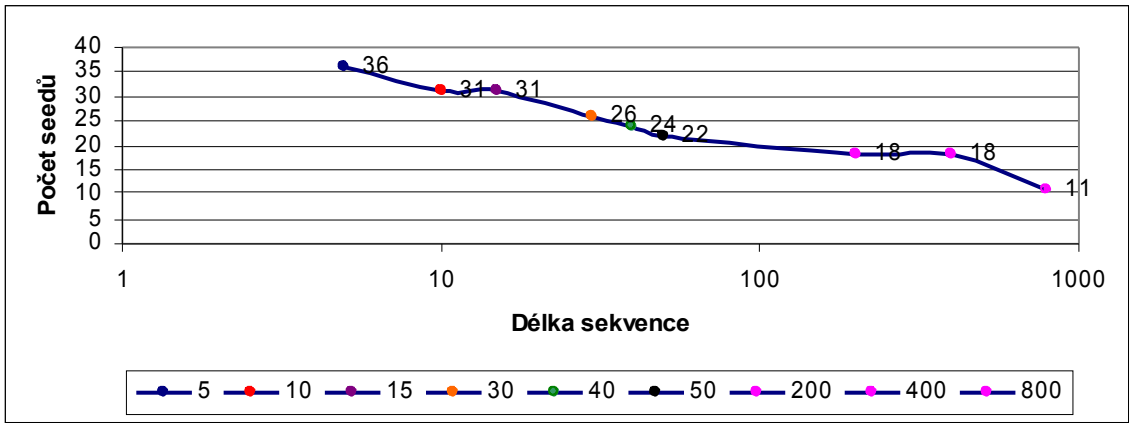
Jedná se opět o poměrně složitý obvod. Bez reseedingu je potřeba k jeho plnému otestování průměrně jeden a půl milionu vektorů.



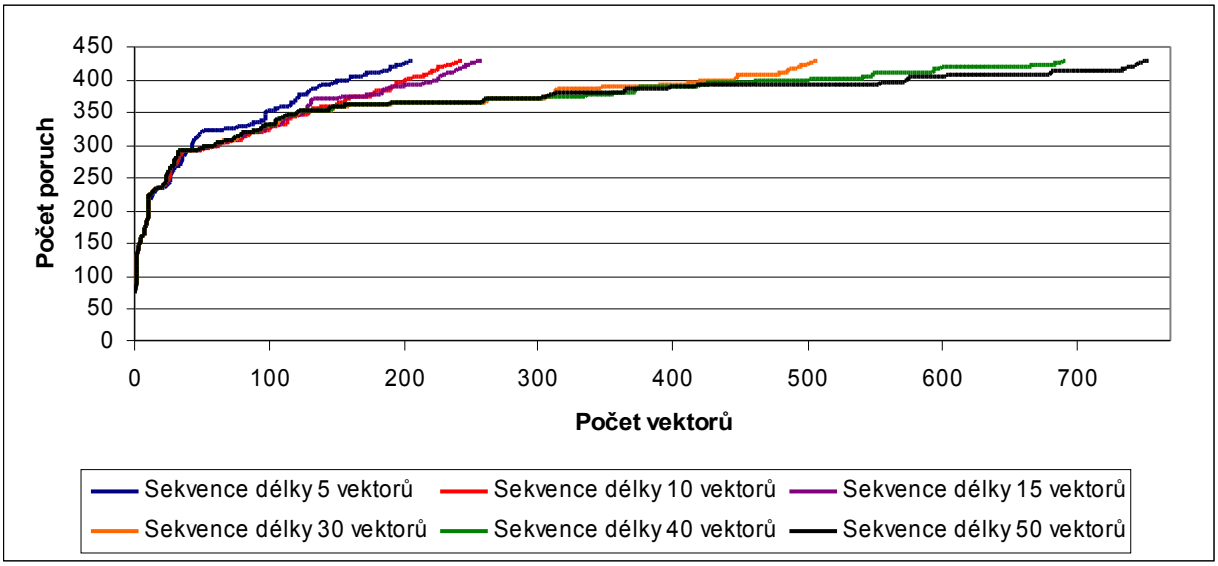
Graf 4.24 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s420)



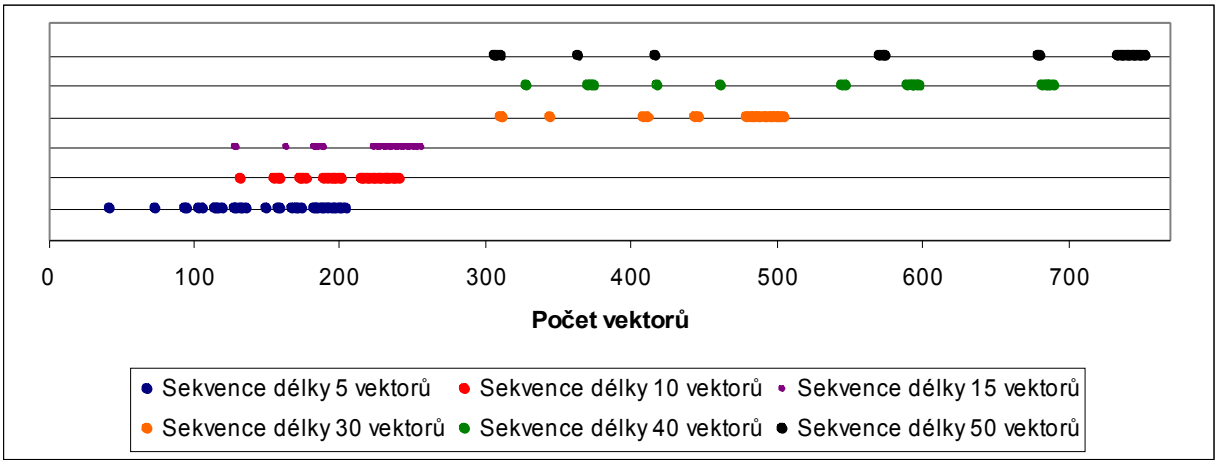
Graf 4.25 Porovnání míst reseedingu dle délky sekvence (obvod s420)



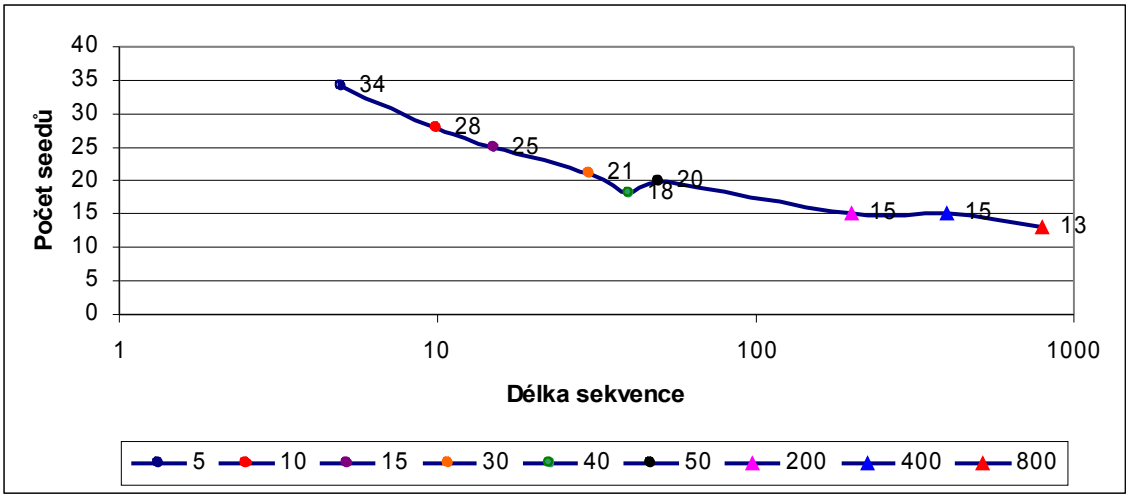
Graf 4.26 Vývoj počtu seedů dle délky sekvence (obvod s420)



Graf 4.27 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod s420)



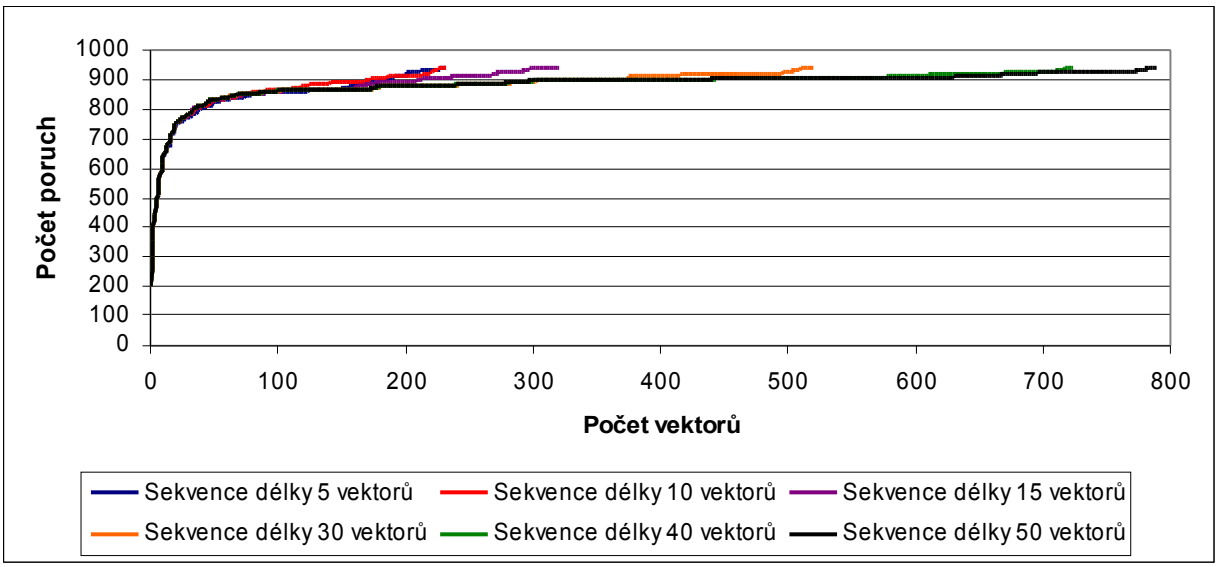
Graf 4.28 Porovnání míst reseedingu dle délky sekvence (obvod s420)



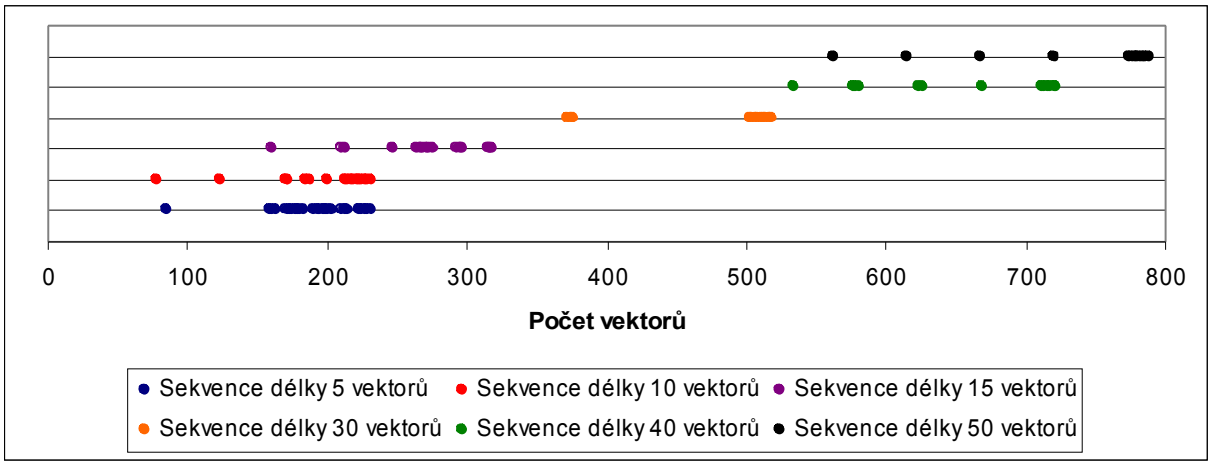
Graf 4.29 Vývoj počtu seedů dle délky sekvence (obvod s420)

4.5.2.4 Výsledky měření pro obvod c880

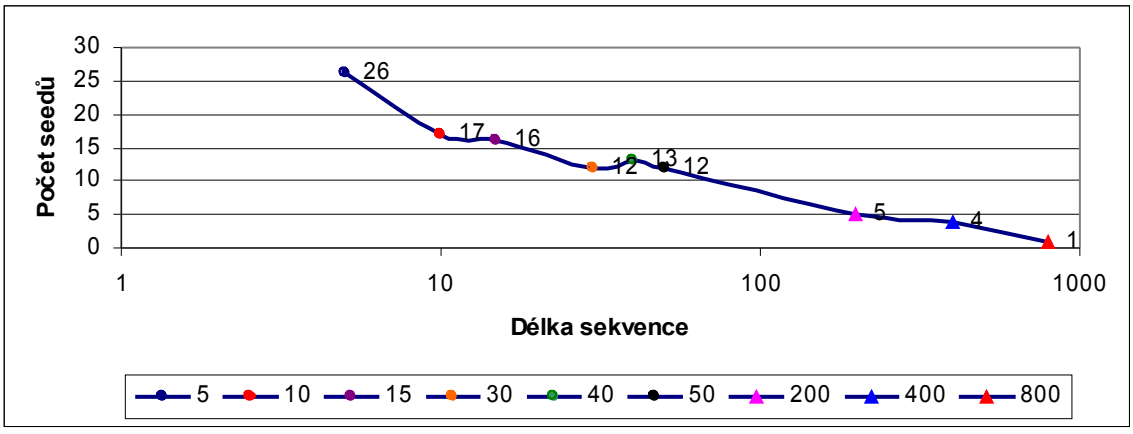
Jedná se o jednoduchý obvod, nicméně bez reseedingu je potřeba k jeho plnému otestování průměrně třináct tisíc vektorů.



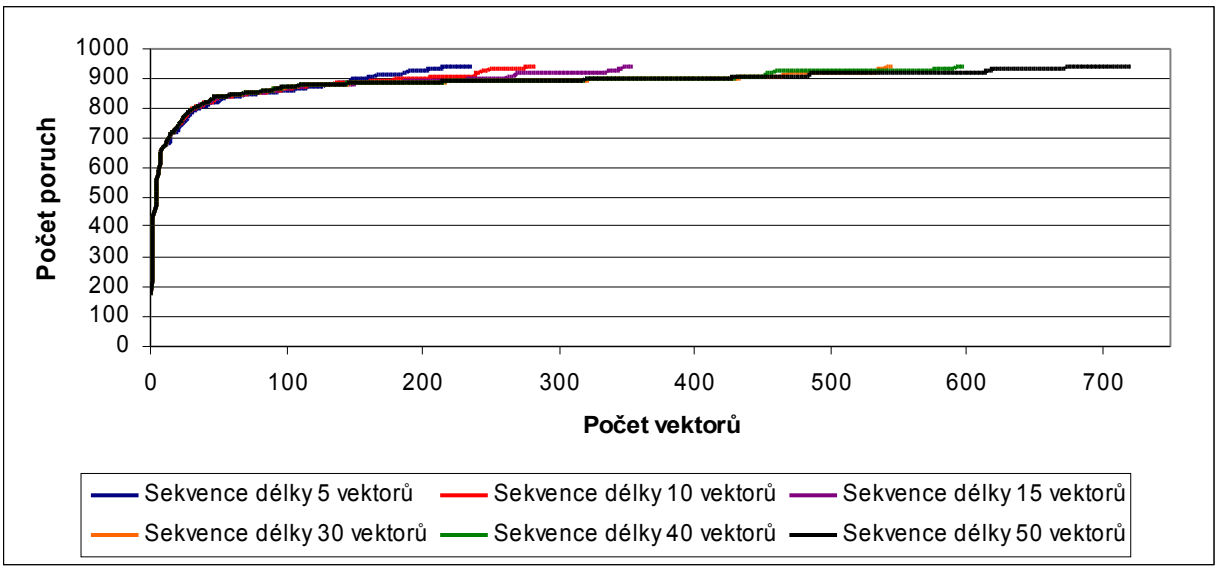
Graf 4.30 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod c880)



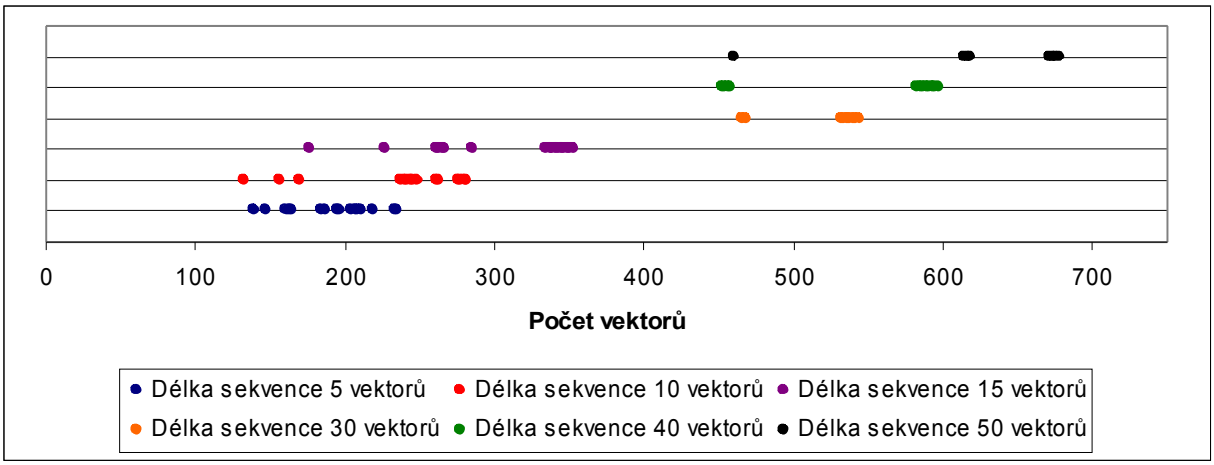
Graf 4.31 Porovnání míst reseedingu dle délky sekvence (obvod c880)



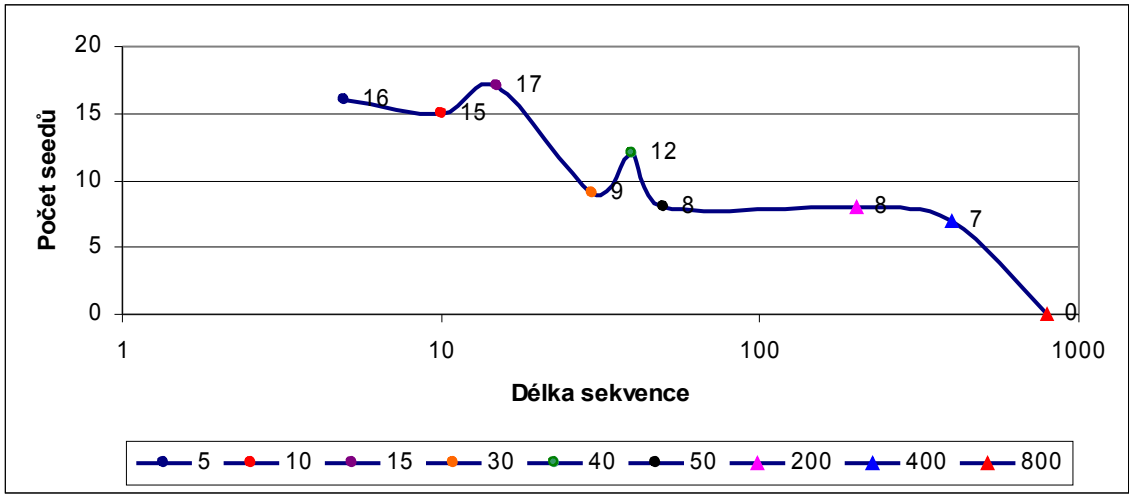
Graf 4.32 Vývoj počtu seedů dle délky sekvence (obvod c880)



Graf 4.33 Porovnání vývoje počtu detekovaných poruch dle délky sekvence (obvod c880)



Graf 4.34 Porovnání míst reseedingu dle délky sekvence (obvod c880)



Graf 4.35 Vývoj počtu seedů dle délky sekvence (obvod c880)

Stěžejními grafy všech výše uvedených měření jsou grafy vyjadřující vývoj počtu seedů dle délky sekvence. Pochopitelně čím delší byla sekvence, tím delší byla i doba testu. Na druhou stranu ve většině případů platilo, že použil-li jsem delší sekvenci, byl zapotřebí menší počet seedů. Tato zjištění mohou být návodem k hledání kompromisu mezi délkou (dobou) testu a množstvím uchovávaných dat.

Provedl jsem též srovnání výsledků, kdy jsem ke generování sekvence použil LZPR v prvém případě a CA(pravidlo 60) v případě druhém. Výsledky porovnání jsou prezentovány v následující kapitole.

5. Porovnání počtu seedů LZPR a CA (prav. 60)

V testech, prezentovaných v předchozí kapitole, jsem použil k získání sekvence kroky LZPR. Pro shodné nastavení parametrů každého jednoho testu, konkrétně seedu, polynomu a délky sekvence jsem provedl obdobné testy s tím rozdílem, že jsem k získání sekvence použil nikoliv kroky LZPR, ale CA a jeho pravidlo 60. I zde bych mohl uvést celou řadu měření a grafů, ale omezím se pouze na prezentaci srovnání vývoje počtu reseedingů pří obou způsobů získaní sekvence.

Data a výsledky všech provedených testů jsou na přiloženém datovém nosiči.

5.1 Výsledky měření pro obvod s838

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	69	61
10	63	56
15	57	56
30	54	42
40	55	48
50	53	32
200	44	14

Tab 5.1

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	72	66
10	64	50
15	56	51
30	58	43
40	57	48
50	52	37
200	47	24

Tab 5.2

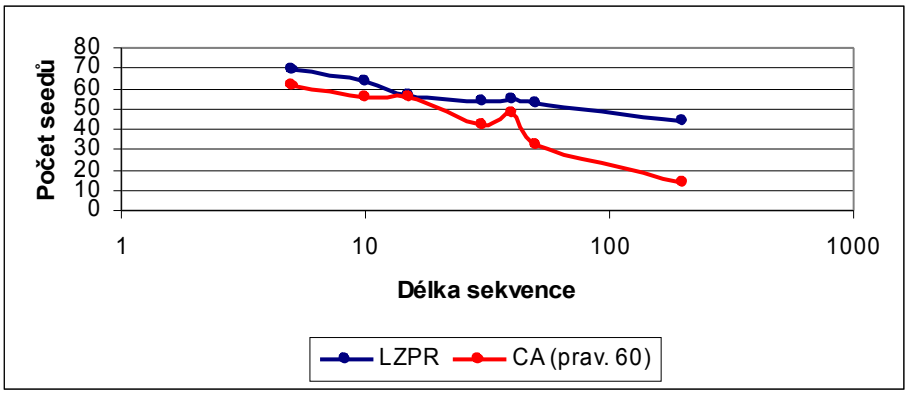
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	73	61
10	66	56
15	64	53
30	58	39
40	55	39
50	50	32
200	46	22

Tab 5.3

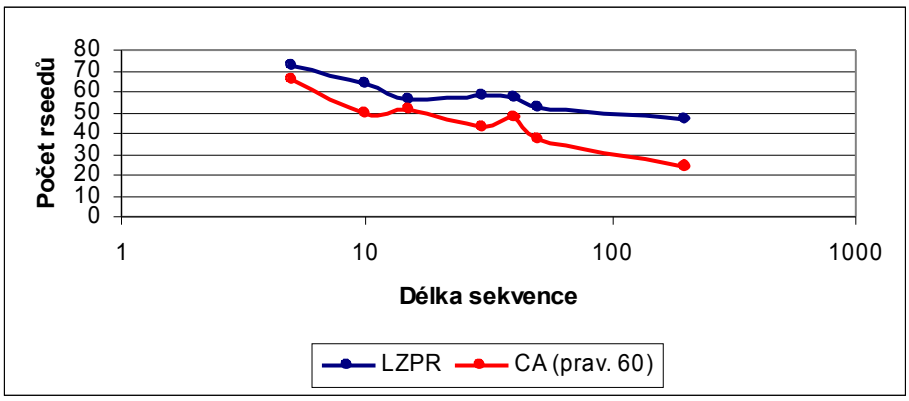
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	67	63
10	59	57
15	61	51
30	51	45
40	53	34
50	50	32
200	46	25

Tab 5.4

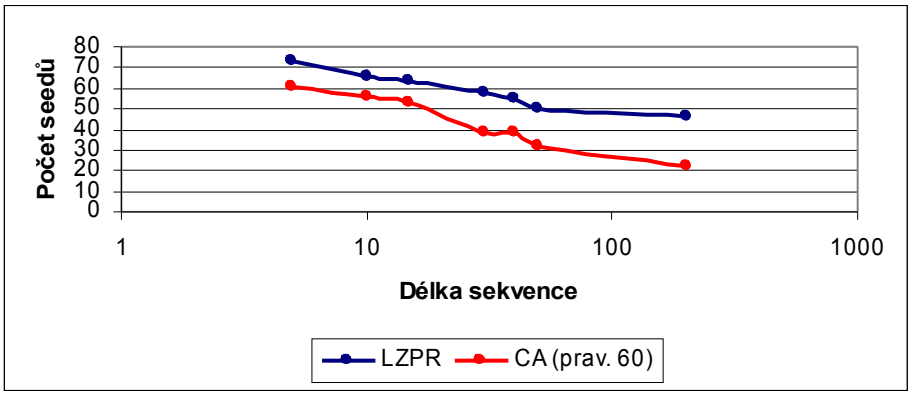
Tab. 5.1 až 5.4 počet seedů pro sekvenci LZPR a CA (60), obvod s838



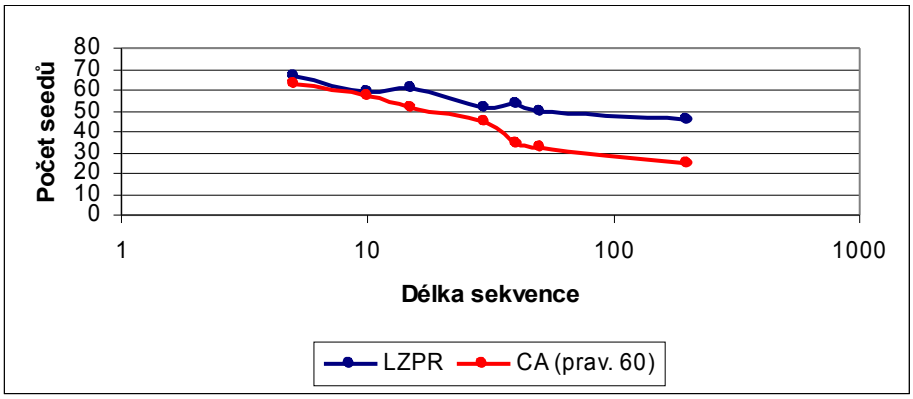
Graf 5.1 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.1) (obvod s838)



Graf 5.2 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.2) (obvod s838)



Graf 5.3 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.3) (obvod s838)



Graf 5.4 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.4) (obvod s838)

5.2 Výsledky měření pro obvod s713

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	40	39
10	27	39
15	28	31
30	23	20
40	19	17
50	17	19
200	9	11

Tab 5.5

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	39	46
10	29	32
15	27	22
30	18	18
40	17	19
50	17	17
200	12	3

Tab 5.6

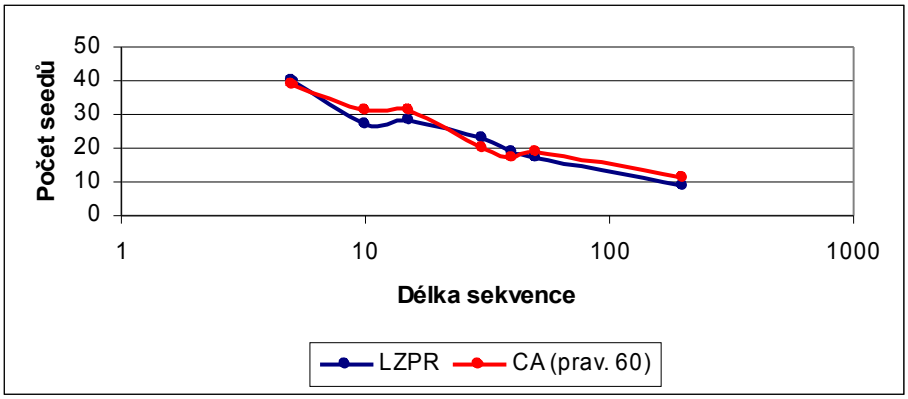
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	30	39
10	29	30
15	24	22
30	16	23
40	18	19
50	19	14
200	10	10

Tab 5.7

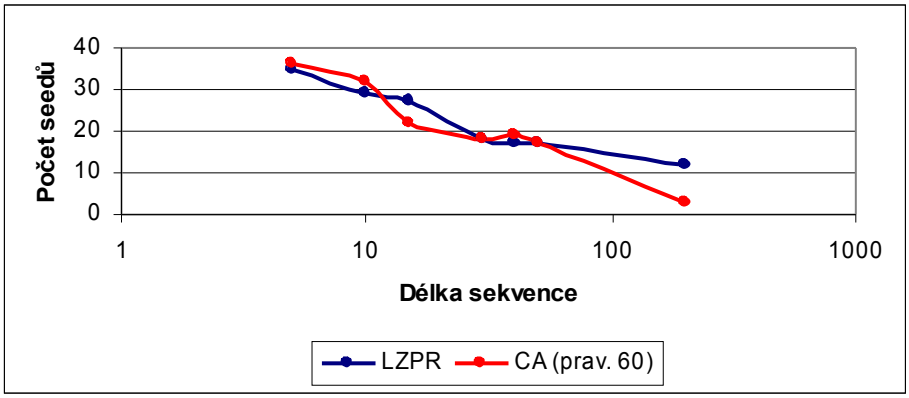
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	36	29
10	31	27
15	25	26
30	20	17
40	22	17
50	16	15
200	9	10

Tab 5.8

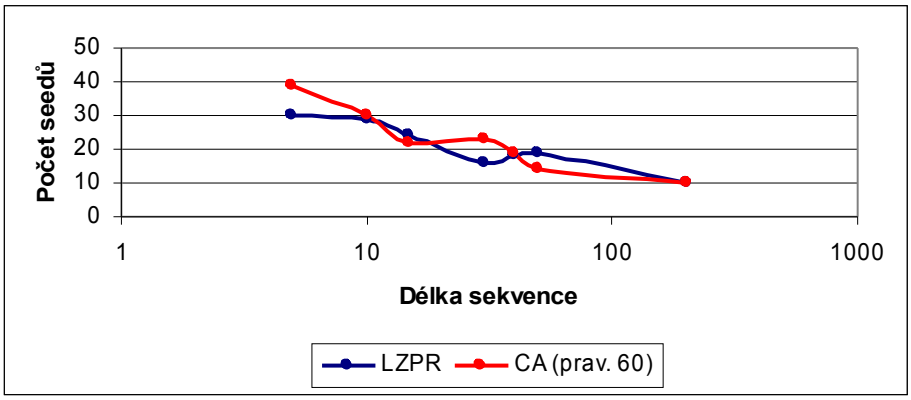
Tab. 5.5 až 5.8 počet seedů pro sekvenci LZPR a CA (60) (obvod s713)



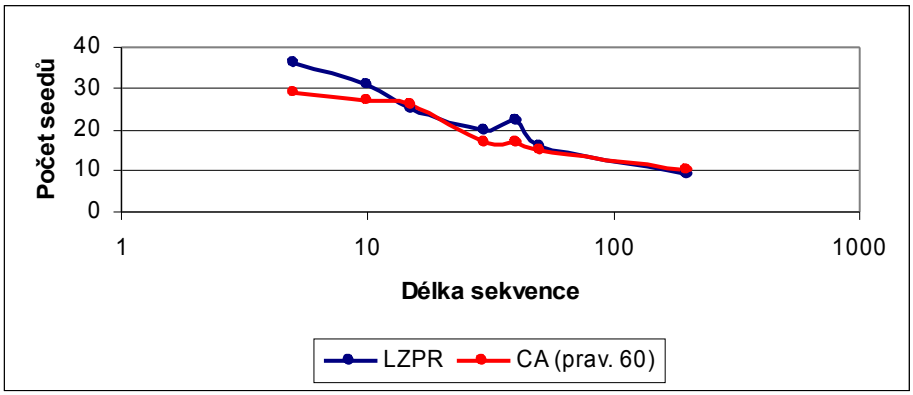
Graf 5.5 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.5) (obvod s713)



Graf 5.6 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.6) (obvod s713)



Graf 5.7 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.7) (obvod s713)



Graf 5.8 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.8) (obvod s713)

5.3 Výsledky měření pro obvod s420

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	36	36
10	31	25
15	31	29
30	26	19
40	24	18
50	22	17
200	18	10
400	18	14
800	11	11

Tab 5.9

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	34	29
10	28	25
15	25	33
30	21	18
40	18	18
50	20	18
200	15	11
400	15	15
800	13	11

Tab 5.10

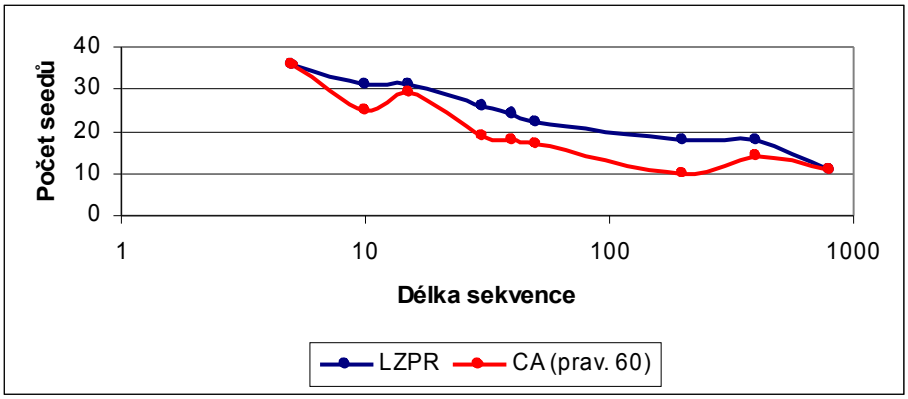
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	34	35
10	27	20
15	25	23
30	24	16
40	21	15
50	21	16
200	17	13
400	17	14
800	15	17

Tab 5.11

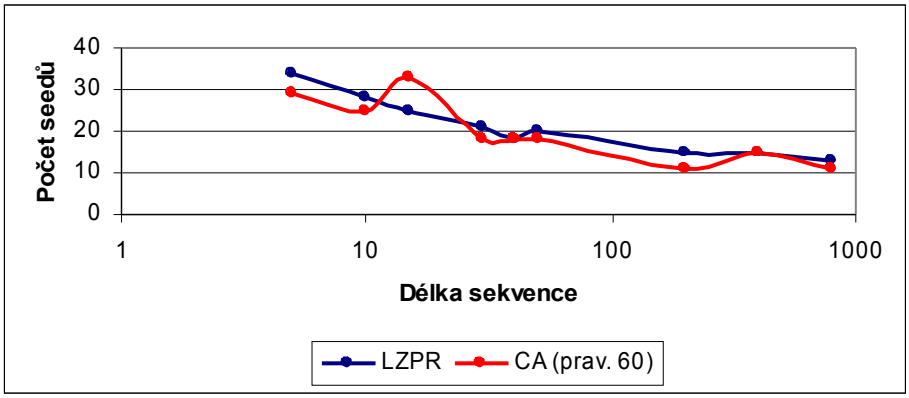
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	35	32
10	30	27
15	27	26
30	25	21
40	19	16
50	20	14
200	15	14
400	15	13
800	13	10

Tab 5.12

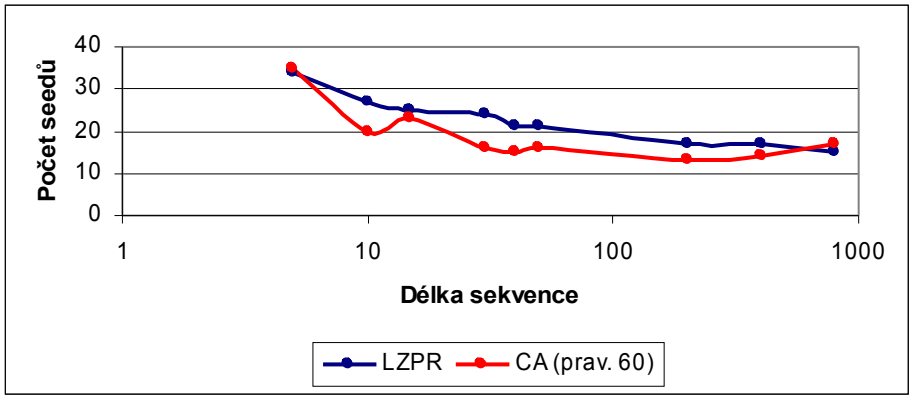
Tab. 5.9 až 5.12 počet seedů pro sekvenci LZPR a CA (60) (obvod s420)



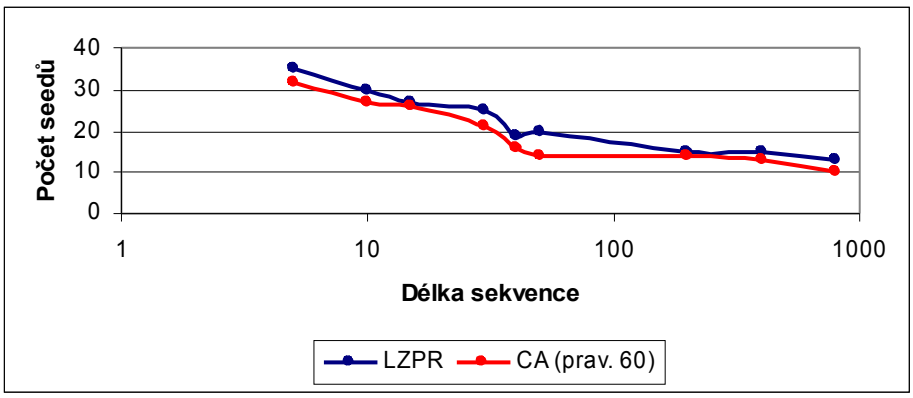
Graf 5.9 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.9) (obvod s420)



Graf 5.10 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.10) (obvod s420)



Graf 5.11 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.11) (obvod s420)



Graf 5.12 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.12) (obvod s420)

5.4 Výsledky měření pro obvod c880

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	23	25
10	17	20
15	19	14
30	13	14
40	11	16
50	11	10
200	4	13
400	4	13
800	2	7

Tab 5.13

délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	24	20
10	16	18
15	14	15
30	14	15
40	11	14
50	11	15
200	5	13
400	5	14
800	0	17

Tab 5.14

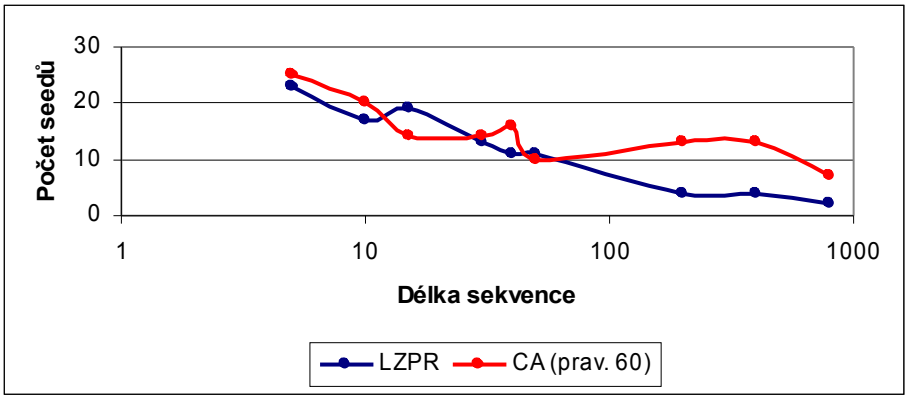
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	22	23
10	16	16
15	15	16
30	10	16
40	15	12
50	11	15
200	4	18
400	3	13
800	1	5

Tab 5.15

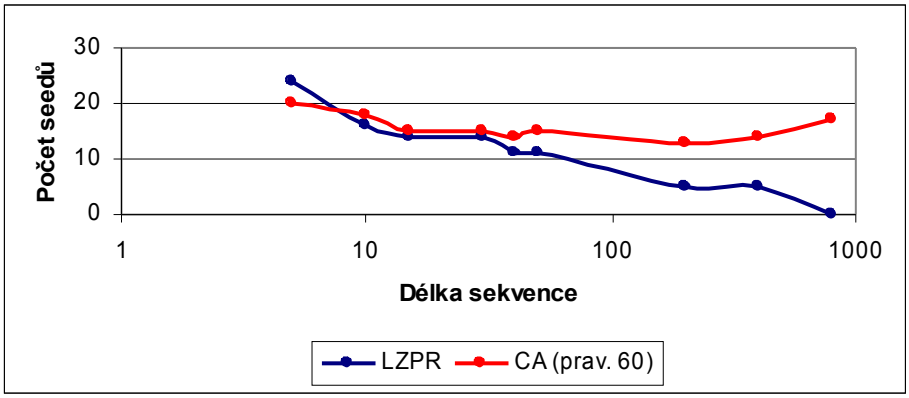
délka sekvence	počet seedů LZPR	počet seedů CA (pravidlo 60)
5	26	18
10	17	20
15	16	22
30	12	11
40	13	13
50	12	15
200	5	16
400	4	12
800	1	13

Tab 5.16

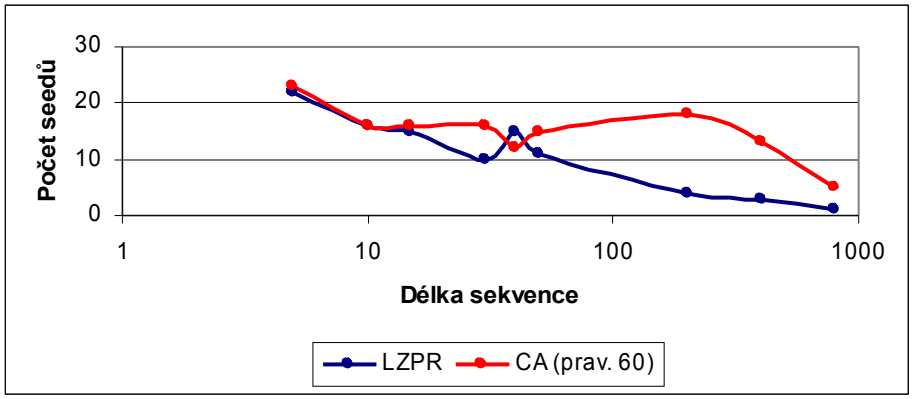
Tab. 5.13 až 5.16 počet seedů pro sekvenci LZPR a CA (60) (obvod c880)



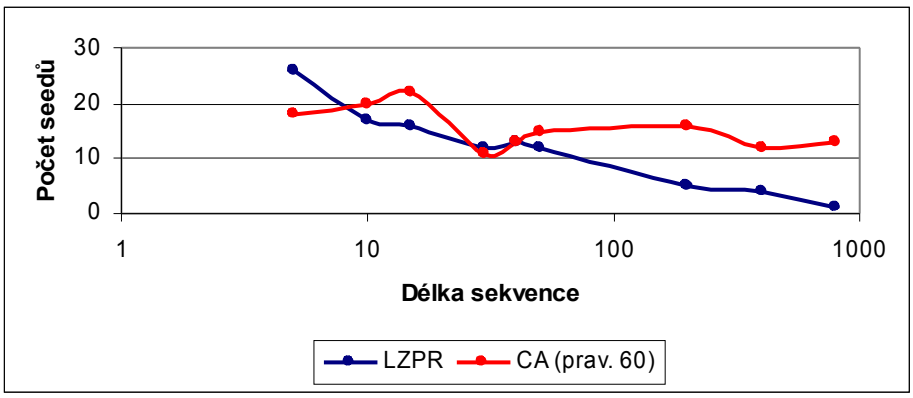
Graf 5.13 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.13) (obvod c880)



Graf 5.14 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.14) (obvod c880)



Graf 5.15 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.15) (obvod c880)



Graf 5.16 Vývoj počtu seedů pro sekvenci LZPR a CA (60) (Tab 5.16) (obvod c880)

5.5 Srovnání výsledků reseedingu s výsledky MPLFSR

V této kapitole je provedeno srovnání výsledků dosažených za pomocí reseedingu (var. 3.) a výsledků uváděných v literatuře [5]. Ty jsou sice získány za pomocí MPLFSR, ale i tak existuje způsob, jak data porovnat. Hodnoty, které porovnávám jsou počet seedů a celkový počet vektorů, který byl použit k detekci všech detekovatelných poruch. Jelikož MPLFSR je silnější nástroj, než jednopolynomiální reseeding, dalo se předpokládat, že výsledky pro jednopolynomiální reseeding budou horší. Ne vždy tomu tak ale bylo, jak je patrné z tabulky 5.17 níže.

K získání testovacích vektorů jsem použil jak LZPR, tak i CA(60) a v obou případech jsem se snažil o to, aby se celkový počet PR vektorů co nejvíce přiblížil počtu referenčnímu. To se mi ne vždy podařilo, ale na druhou stranu i výsledky s menším počtem použitých vektorů byly v několika případech lepší, nebo srovnatelné s výsledky referenčními. Abych ale mohl vyvozovat konkrétní závěry, bylo by potřeba provést mnohem více testů, nicméně i tyto výsledky jsou zajímavé.

MPLFSR				reseeding (var. 3.)			
				LZPR		CA(60)	
bench	počet PR vektorů	počet seedů	počet seedů (použita komprimace)	počet PR vektorů	počet seedů	počet PR vektorů	počet seedů
s5378	1000	135	85	1110	154	1051	141
	2000	97	70	1828	89	1871	87
	5000	51	44	3340	62	4931	41
	10000	35	32	7900	29	7293	28
s9234.1	1000	313	152	1185	349	1148	350
	2000	284	139	2047	280	2140	283
	5000	231	116	5225	213	4560	218
	10000	196	107	8350	180	9447	168
13207.1	1000	398	196	1213	436	1241	451
	2000	222	171	1838	369	1799	368
	5000	247	135	4608	235	4754	226
	10000	174	105	9000	148	8773	153

Tab. 5.17 Srovnání výsledků reseedingu s výsledky MPLFSR

5.6 Shrnutí

Nad každou z variant algoritmu jednopolynomiálního reseedingu jsem provedl řadu testů a vybrané výsledky prezentoval v dílčích podkapitolách. Postupoval jsem od nejjednodušší varianty reseedingu ke složitějším tak, jak jsem získával nové poznatky. Nakonec třetí varianta reseedingu dávala nejlepší výsledky.

Jelikož v úvodu zmiňuji i celulární automat a protože sekvenci vektorů lze získat i s jeho pomocí, rozhodl jsem se provést ještě srovnání pro sekvence vektorů získané pomocí kroků LZPR a sekvence získané dle předpisu pravidla 60 celulárního automatu.

Pro nejhůře PR testovatelný obvod s838 byla sekvence dle pravidla 60 v detekci poruch mnohem úspěšnější, než sekvence dle LZPR, což je vidět na počtu seedů, viz. 5.1 . V méně složitých obvodech byly rozdíly nepatrné a u jednoduchého obvodu (c880) byly výsledky dle pravidla 60 mnohem horší.

Bohužel výpočetní aparát mi nedovoloval testovat složitější obvody ale i tak jsem se potýkal s náročností testů. Například získání dat pro graf 4.10 trvalo sto hodin čistého času běhu počítače (každé měření trvalo hodinu), u grafu 4.11 měření trvalo hodin šedesát. Myslím si ale, že data jsou zajímavá a bude případně možné je použít jako podklad pro další práci.

6 Implementace

V této kapitole se zaměřuji na stručný popis implementace, strukturu vstupních a výstupních dat a popisuji kroky vedoucí k urychlení testování.

6.1 Program

Úspěšně jsem využíval svého programu napsaného v jazyce Java 1.6 v kombinaci s programem Atalanta simulujícím testované obvody. Program Atalanta, jehož bližší popis je uveden v příloze A, byl spouštěn jako externí proces s příslušnými parametry ve formě bat souboru. Pro zajímavost dodávám, že celkově byl program Atalanta spuštěn nejméně deset milionkrát.

Veškeré parametry algoritmů byly načítány z konfiguračního souboru. Parametry jako počet detekovatelných poruch a počet primárních vstupů byly načítány ze souboru vygenerovaného Atalantou. Je zcela zřejmé, že vstupní data musí splňovat určitou formu, takže jsou provedeny i validace a v případně nevalidních dat je uživatel informován. Celý program i dokumentace k němu je uložen na přiloženém datovém nosiči.

6.2 Vstupní data

Nezbytnými vstupními daty jsou bench soubor, který reprezentuje simulovaný obvod, označení obvodu (s838, s713, ...), počet vstupů obvodu (parametr *iv*), počet detekovatelných poruch (parametr *d_faults*), celkový počet poruch (parametr *faults*), seed v binární reprezentaci a polynom, též v binární reprezentaci. V některých případech se uvažují i další parametry, například délka sekvence.

Struktura dat bench souboru je uvedena níže (komentáře nejsou v souboru uvedeny):

gates: 390	-- počet hradel v obvodu
iv: 67	-- počet primárních vstupů
ov: 34	-- počet primárních výstupů
i_patterns: 150	-- počet testovacích vektorů před kompresí
patterns: 100	-- počet testovacích vektorů po kompresi
faults: 857	-- celkový počet poruch
d_faults: 857	-- celkový počet detekovatelných poruch
r_faults: 0	-- celkový počet nedetekovatelných (redundantních) poruch, zde výjimečně nula

Výše uvedené hodnoty parametrů platí pro obvod s838, ale pochopitelně se mění dle vybraného obvodu. Struktura dat ale zůstává shodná pro všechny simulované obvody. Stěžejními parametry jsou *iv*, *faults* a *d_faults*, viz. dále.

K detekci poruch se používají testovací vektory. Délka takovéhoto vektoru musí být shodná s hodnotou parametru *iv*. Každý takovýto vektor detekuje určité poruchy v obvodu a vždy chci dosáhnout toho, aby pomocí sady vektorů byly detekovány všechny detekovatelné poruchy, jejichž počet je dán hodnotou parametru *d_faults*.

6.3 Výstupní data

Výstupními daty je několik textových souborů, které zachycují celý průběh algoritmu. Nejdůležitější z nich je soubor, v němž jsou zapsány všechny testovací vektory, nad nimiž se simuluje test. Lze ověřit, že tato konečná sada testovacích vektorů beze zbytku detekuje všechny detekovatelné poruchy, pokud byla získána pomocí algoritmu, od nějž vyžaduji aby jím generované vektory detekovaly v rozumně krátkém čase všechny detekovatelné poruchy.

Další důležité soubory jsou soubory v nichž jsou zapsány počty detekovaných poruch v krocích algoritmu, případně i místa, kde došlo k reseedingu, eventuálně i další parametry.

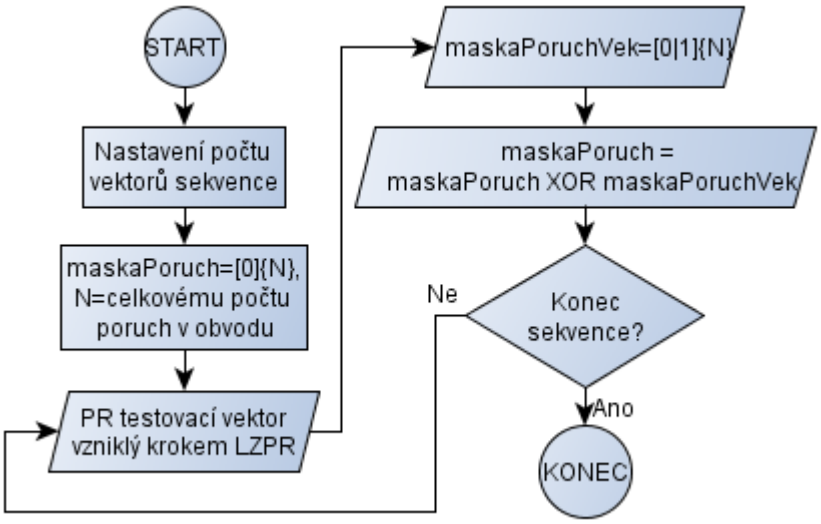
Na základě dat z těchto souborů byly realizovány tabulky, grafy a konečně vyhodnocování i závěry.

6.4 Urychlení testování

Jelikož testy byly pro složitější obvody časově velmi náročné, hledal jsem způsob, jak je urychlit. Řekněme, že chci vygenerovat testovací sekvenci o padesáti vektorech, čili provedu padesátkrát krok LZPR, ale zároveň mě zajímá, kolik poruch je detekováno po každém kroku. První způsob je ten, že vždy otestuji celou sekvenci, což znamená že padesátkrát otestuji sekvenci o narůstajícím počtu vektorů. V prvním kroku otestuji sekvenci o jednom vektoru, v dalším kroku sekvenci o dvou vektorech, atd. až konečně v padesátém kroku otestuji celou sekvenci o padesáti vektorech. Tento způsob testování je možný pro malé počty vektorů, ale pokud se jedná o tisíce, nebo desetitisíce vektorů, je test velmi pomalý.

Druhý, rychlejší a nakonec výhradě mnou používaný způsob je ten, že si od každého vektoru vezmu masku poruch, což je binární řetězec délky rovnajíce se počtu detekovatelných poruch. Tento řetězec má na příslušných pozicích jedničku, pokud vektor detekuje danou poruchu, v opačném případě je na dané pozici nula. Počet jedniček v dané masce tedy udává počet poruch, které vektor detekuje. Abych zjistil, kolik *N* vektorů detekuje poruch, stačí provést *N-1* krát xor masek poruch.

Tento způsob testování je nesrovnatelně rychlejší. Schema algoritmu vyhodnocování počtu poruch v sekvenci je pro představu uvedeno níže.



Schema 6.1. Schema algoritmu vyhodnocování počtu poruch v sekvenci

7. Závěr

Celá problematika ohledně testování obvodů je velmi složitá a existuje celá řada algoritmů, které mohou sloužit jako generátory testovacích vektorů. Já si vybral a důkladně otestoval reseeding. Reseeding byl realizován ve třech rozdílných variantách, které se lišily v podmínce, kdy má vlastní reseeding nastat. Ve všech variantách byl reseeding použit pro kompresi testovacích vektorů.

Co se variant reseedingů týče, lze říci, že jsem uplatnil inkrementální postup. Každá následující varianta byla totiž realizována na základě dat, výsledků a zkušeností získaných z varianty předchozí a až teprve třetí varianta dávala nejlepší výsledky. Pro poslední variantu jsem též vyzkoušel dva způsoby generování sekvence testovacích vektorů a to sice pomocí kroků LZPR a dle pravidla 60 celulárního automatu. Porovnání výsledků obou variant je též prezentováno ve 4. kapitole.

Pro každou variantu algoritmů jsem provedl řadu testů pro různě nastavené parametry a pro různě složitě testovatelné obvody a vybrané výsledky prezentoval v příslušných podkapitolách. Z uvedených výsledků je jasně vidět pokrytí poruch, časová i prostorová náročnost. Veškerá ostatní data, testy a výsledky testů, které nejsou uvedeny přímo v textu jsou na přiloženém datovém nosiči.

Výběr obvodů, počet i náročnost testů jsem tedy volil s ohledem na parametry mého výpočetního aparátu. Troufám si ale říct, že ze získaných dat lze činit závěry, a že data jsou zajímavá a bude případně možné je použít jako podklad pro další práci.

Použitá literatura a zdroje

[1] P. Galaktionov, "Syntéza generátorů testovacích obvodů pomocí genetických algoritmů", Master's thesis, FEL, ČVUT, 2006.

[2] A. Pluháček, "Přednášky z předmětu Bezpečnostní a jiné kódy", 2007, http://service.felk.cvut.cz/courses/X36BJK/.

[3] P. Fišer and H. Kubátová, "Pseudorandom Testability, Study of the Effect of the Generator Type", Acta Polytechnica, Vol. 45, No. 2, August 2005, CVUT, ISSN 1210-2709, pp. 47-54

[4] Petr Šafář, "Návrh nelineárních celulárních automatů", Master's thesis, FEL ČVUT, 2008

[5] Sybille Hellebrand, Janusz Rajski, Steffen Tarnick, Srikanth Venkataraman, Bernard Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers", Inst. for Comput. Structures, Siegen Univ., pp. 223-233, Feb. 1995

[6] Srikanth Venkataraman, Janusz Rajski, Sybille Hellebrand, Steffen Tarnick, "An Efficient BIST Scheme Based on Resseding of Multiple Polynomial Linear Feedback Shift Registers", Proceedings ACM/IEEE International Conference on Computer-Aided Design (ICCAD'93)

[7] Bernd Könemann, "LFSR-Coded Test Patterns for Scan Designs", Proc. Europ. Test. Conf. , Munich, 1991, pp. 237-242

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

[9] H.K. Lee and D.S. Ha, "An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation", Proc. Int. Test Conf., pp. 946-955, October 1991.

[10] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits, "IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems", Vol. 15, pp. 1048-1058, September 1996.

[11] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan", Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985.

[12] F. Brglez, D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989.

[13] http://www.wikipedia.org/

Příloha A

Program Atalanta

Program Atalanta simuluje fyzické obvody i testy nad těmito obvody. Bez tohoto nástroje by celá diplomová práce ani nemohla vzniknout. Bližší popis, tutoriál i zdrojové kódy jsou k dispozici na webových stránkách na adrese: http://service.felk.cvut.cz/vlsi/prj/Atalanta-M/. Pro představu níže uvádím několik příkladů, jak program použít.

atalanta-M -t c432.pat -W 1 c432.bench
Generates test patterns for c432.bench, writes them to c432.pat.

atalanta-M -t c432.pat -W 1 -F c432.flt c432.bench
Generates test patterns for c432.bench, writes them to c432.pat. The complete fault list is written to c432.flt.

atalanta-M -t c432.pat -W 2 c432.bench
Generates test patterns for c432.bench, writes them to c432.pat. The correct CUT responses are computed as well.

atalanta-M -D 1 -t c432.pat -W 2 c432.bench
Generates test patterns for c432.bench, writes them to c432.pat. One test pattern is produced for each fault. Don't cares are present in the test. The correct CUT responses are computed as well.

atalanta-M -t c432.pat -f c432.flt -W 1 c432.bench
Generates test patterns for c432.bench, writes them to c432.pat. Only faults specified in c432.bench are considered.

atalanta-M -t c432.pat -P c432.rep -m c432.mask -W 1 c432.bench
Generates test patterns for c432.bench, writes them to c432.pat. The report file is written to c432.rep, the fault mask (after applying the whole test to CUT) is written to c432.mask.

atalanta-M -S -t c432.pat -P c432.rep c432.bench
Simulates vectors for c432.pat and writes the results to c432.rep.

atalanta-M -S -t c432.pat -P c432.rep -U c432.ud -v c432.bench
Simulates vectors for c432.pat and writes the results to c432.rep. All the undetected faults are written to c432.ud.

atalanta-M -S -t c432.pat -P c432.rep -m c432.msk c432.bench
Simulates vectors for c432.pat and writes the results to c432.rep. The fault mask is written to c432.msk

atalanta-M -l 100040000001 765177704700 1000 -U c432.flt -v -P c432.rep c432.bench
Simulates 1000 LFSR vectors for c432.pat and writes the results to c432.rep. The LFSR generating polynomial is 100040000001 (in octal), the seed 765177704700 (in octal). The undetected faults list is written to c432.flt.

Příloha B

Obsah přiloženého CD

- všechny zdrojové kódy programu
- dokumentace k programu (java doc)
- data, grafy a tabulky pro všechna provedená měření
- celý text diplomové práce ve formátu odt a pdf