**ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE**

Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

# Implementace prostředků pro vestavěnou diagnostiku v FPGA

Vedoucí práce : Fišer Petr Ing., Ph.D.

Studijní program : Elektrotechnika a informatika strukturovaný magisterský

Obor : Informatika a výpočetní technika

Květen 2008

## Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti použití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o autorském právu, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 15.května 2008 …….

## Abstrakt:

Tato práce se zabývá implementací vestavěných diagnostických prostředků v jazyce VHDL.

Obvod je zadán pomocí netlistu. Vygenerován je syntetizovatelný VHDL kód obvodu, spolu s vestavěnou diagnostikou. Metoda provedení testu obvodu je k dispozici dvěmi způsoby a to test-per-scan, popřípadě test-per-clock. Technikou reseedingu je za pomoci několika pomocných programů realizován algoritmus pro co nejlepší pokrytí poruch, dle uživatelem zadaných parametrů. Popřípadě je nabídnuta možnost manuálního zadání testovacích vektorů pro reseeding, pro které je k dipozici také technika založená na multi-polynomiálním LFSR.

Výsledkem je tak několik programů generujících VHDL obvodu spolu s BISTEM, těmito rozličnými metodami, k nim vytvořené dokumentace a manuály pro jejich použití.

# Obsah

# 1. Úvod

## 1.1 Přesné znění pokynů práce

Vytvořte nástroj pro integraci prostředků pro vestavěnou diagnostiku (BIST) do obvodu zadaného netlistem. Konkrétně se jedná o implementaci BIST techniky "reseeding", případně některých dalších. Uvažujte test-per-clock i test-per-scan BIST.

Výstupem bude syntetizovatelný VHDL kód reprezentující funkční obvod se začleněným BISTem. Nástroj odzkoušejte na zkušebních obvodech, naimplementujte tyto obvody s BISTem v FPGA. Otestujte funkčnost obvodů s BISTem, tj. jsou-li schopné se otestovat a objevit případnou poruchu.

## 1.2 Cíle práce a požadavky na implementovaný systém

### 1.2.1 Implementační cíle a požadavky

Cílem práce je naimplementovat obvody reprezentující vestavěné diagnostické prostředky v jazyce VHDL a tyto BISTy vygenerovat spolu se zadaným obvodem. Tento obvod bude programu zadán pomocí netlistu a program jej převede do jazyka VHDL a připojí k němu vygenerovaný BIST. Celý tento obvod pak bude plně syntetizovatelný a funkční, což vyzkouším na obvodech FPGA.

Generovaný BIST bude implementován dvěma metodami pro provedení testu, test-per-clock i test-per-scan. Technikou reseedingu pak bude realizován algoritmus pro co nejlepší pokrytí poruch, dle uživatelem zadaných parametrů. Popřípadě je nabídnuta možnost manuálního zadání testovacích vektorů pro reseeding, pro které je k dispozici také technika založená na multi-polynomiálním LFSR.

### 1.2.2 Dokumentační cíle a požadavky

K implementovaným programům a prostředkům bude vytvořena dokumentace, pro jejich snadné použití a pochopení funkce. K vytvořeným programům bude přiložen manuál popisující práci s nimi pro správnou funkčnost a využití.

### 1.2.3 Osobní cíle

Zlepšit své znalosti v oblasti vestavěných diagnostických prostředků a diagnostice a spolehlivosti obecně. Jelikož jsem se této oblasti věnoval již ve své bakalářské práci a týmovém projektu v předešlých semestrech, mohl jsem tyto zkušenosti nyní využít.

Zdokonalit se v používání návrhových systémů a prostředků od renomovaných firem (Xilinx, Mentor Graphics) a přípravků s programovatelnými obvody.

Zdokonalit se a procvičit ve znalosti programovacích jazyků jazyka C a VHDL.

# 2. Úvod do diagnostiky a obecná charakteristika implementovaných prostředků

Tato kapitola poskytuje úvod do diagnostiky poruch logických obvodů a dále se věnuje obecně principům a konstrukcím diagnostických prostředků implementovaných v této práci.

## 2.1 Úvod do diagnostiky

Podle toho, zda je testovaný obvod v daném okamžiku schopen vykonávat předepsanou funkci, rozlišujeme dva stavy: poruchový a bezporuchový. Základní úlohou diagnostiky je rozpoznání poruchového a bezporuchového stavu u testovaného obvodu. Výsledkem testu je tedy pouze jednobitová informace udávající stav testovaného obvodu v reálném čase. Pomocí testu se nezjišťuje typ poruchy a ani kolik poruch se v testovaném obvodu vyskytuje.

### 2.1.1 Poruchy

Porucha je jev, při kterém objekt, v našem případě obvod, přestává plnit svou předepsanou funkci. Naproti tomu chyba je rozdíl mezi správnou a skutečnou hodnotou. Poruchy se dají rozdělit na fyzické a logické. Mnoho různých typů fyzických poruch lze vyjádřit jednou logickou poruchou, avšak popis pomocí logické poruchy je určité zjednodušení, protože při testování nelze brát v úvahu všechny poruchové stavy, které se mohou v testované jednotce vyskytnout.

Základní rozdělení poruch:

- fyzikální porucha (physical fault) = defekt (defect) – nežádoucí fyzikální jev, který v obvodu nastal (např. přerušení vodiče, zkrat, průraz P-N přechodu apod.)
    - lehko detekovatelné (easy-to-detect) fyzikální poruchy – poruchy, které mohou být šířeny na výstup obvodu přiložením mnoha testovacích vzorků na vstup obvodu
    - těžko detekovatelné (hard-to-detect) fyzikální poruchy – poruchy, které mohou být šířeny na výstupy obvodu pouze použitím několika málo testovacích vzorků přiložených na vstupy obvodu
- logická porucha (fault) – model fyzikální poruchy, jehož zápis je prováděn prostředky obvyklými pro logické obvody

Dělení poruch dle závislosti na čase:

- stálé – poruchy neměnné v čase
- nestálé – poruchy, jenž se mění v čase
    - přechodné
    - náhodné

Dělení poruch dle jejich výskytu:

- jednotlivé (single faults)
- násobné (multiple faults)

Základní kategorie logických poruch:

- trvalé poruchy (stuck-at-faults) - označované jako trvalé poruchy typu t0/t1, poruchami typu t lze modelovat celou řady logických poruch uvnitř logických členů obvodu (např. přerušení vodiče, zkrat vodiče na zdroji napájecího napětí, přerušení napájecího vedeni, přerušení uzemnění atd. ), modelování fyzikálních poruch je silně závislé na použité technologii, některé takové poruchy pak nelze poruchami typu t vůbec popsat, je proto nutné použít jiné modely fyzikálních poruch, viz dále
  - trvalá t0 – porucha projevující se jako trvalá 0 na příslušném vodiči, lze ji modelovat připojením na konstantní zdroj trvalé nuly
  - trvalá t1 - porucha projevující se jako trvalá 1 na příslušném vodiči, lze ji modelovat připojením na konstantní zdroj trvalé jedničky

- zkraty, přemostění (shořte, bridges faults) - popisují zkraty mezi vodiči, pro jejich popis lze použít fiktivní logický člen, jehož typ závisí na typu logických členů, jejichž vývody jsou zkratovány, mohou zapříčinit chování obvodu jako paměť,

- přerušení – poruchy zavedené především pro obvody kategorie CMOS, kdy je přerušena komplementární funkce tranzistorů s kanály P a N

- citlivost na vzorek – vznikají v důsledku vysoké hustoty integrace aktivních součástek na čipu, chyba se na výstupu objeví pouze při určité kombinaci vstupních dat

- poruchy zpoždění - jsou těžko detekovatelné, závisí na jmenovité rychlosti obvodu
  - poruchy zpoždění přes logický člen (gate delay)
  - poruchy zpoždění přes kombinační část (path delay)

- poruchy programovatelných polí – průsečíkové poruchy na průsečících propojení a další specifické poruchy způsobené rozličnou strukturou programovatelných polí.

### 2.1.2 Diagnostika a testování

Testem se rozumí soustavy vstupních a jim přiřazených výstupních hodnot, které slouží k odhalení poruch. Počet poruch pokrytých testem vyjadřuje diagnostické pokrytí testu. Pokud test pokrývá všechny detekovatelné poruchy nazývá se test úplný. Test obsahující nejmenší možný počet testovacích vektorů pak test minimální, což je náročné a drahé k nalezení, proto se v praxi nepoužívá. Dalším typem testu je test triviální, který tvoří soustavu všech možných testovacích vektorů, Je to test úplný skládající se ze všech kombinací vstupních proměnných

Při testování se vychází ze základního schématu, nezávislém na zvoleném způsobu testování, viz obrázek 2.1.


Obr. 2.1 Základní schéma diagnostických testů

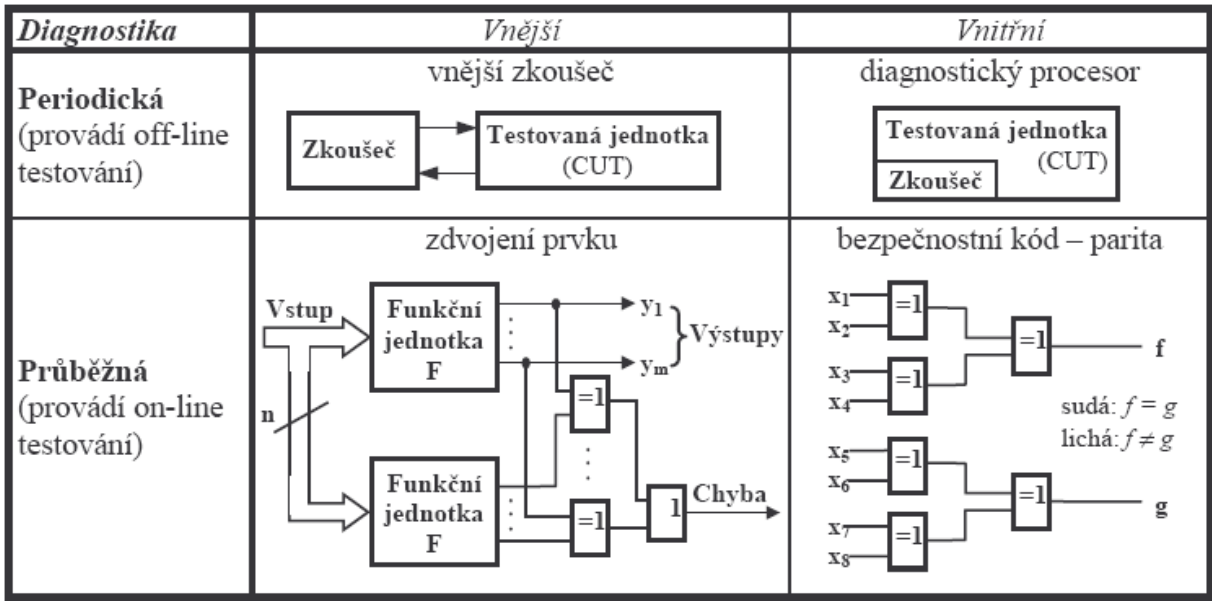
Dělení diagnostiky podle doby provádění diagnostiky:

- periodická
  - provádí se v pravidelných intervalech (např. v přestávkách mezi výpočty), neboť po dobu testů musí být výpočet přerušen, aby mohl být proveden test,
  - provádí off-line testování
- průběžná
  - založena na nepřetržitém sledování a vyhodnocování signálů, které udávají správnost funkce určité testované jednotky během její normální činnosti, provádí
  - on-line testování

Dělení diagnostiky podle umístění zkoušeče vůči testované jednotce:

- vnější
  - používá vnější zkoušeč
  - nevýhodou je velmi vysoká cena
  - v mnoha případech představují jediné možné řešení, kdy zkoumaný obvod není opatřen vnitřní diagnostikou a není možné do něho tyto prostředky začlenit
- vnitřní
  - používá vnitřní zkoušeč
  - každá testovaná jednotka má svůj vestavěný zkoušeč, který komunikuje s okolím pomocí velice jednoduchých signálů (start/stop, dobře/špatně)
  - nevýhodou je potřeba více logických členů či větší spotřeba plochy čipu, určité snížení výkonnosti obvodu
  - výhodou je nižší cena oproti testování pomocí vnějšího zkoušeče
  - do této skupiny patři BIST, kterým se budeme dále zabývat

Shrnutí výše uvedených typů diagnostiky a příklady použití:




Obr. 2.2 Formy diagnostiky a jejich realizace

Z metodického hlediska existují 2 základní strategie testování

- strukturní - jejich sestavení vyžaduje podrobnou analýzu testované jednotky, vytvoření modelu všech poruch, které se v testované jednotce mohou vyskytnout a odvození kroku nebo skupiny kroků pro každou poruchu, při jejich sestavování je výhodné převést sekvenční obvod na kombinační (rozpojením zpětných vazeb), využívají strukturovaného návrhu pro snadnou diagnostiku

výhody:

- převádí problém testování sekvenčních obvodů na testování obvodů kombinačních
- použití jednoduchého obvodu pro generování testovacích vzorků – TPG (Test Pattern Generator)
- snazší simulace poruch

nevýhody

- vyšší odběr elektrické energie
- navržený obvod zabírá větší plochu
- snižuje se rychlost celého obvodu

- funkční - jejich sestavování nevyžaduje podrobnou znalost jejich struktury, ale vychází pouze z popisu jejich funkce, bez znalosti struktury nelze vycházet z modelu fyzikálních poruch, proto je potřeba vytvořit model chování poruch, používají se především tam, kde struktura testovaného obvodu skutečně není známá

výhody

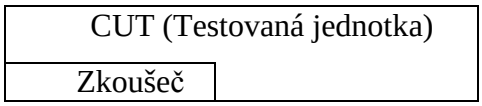
- sestavování funkčního testu je nezávislé na použité technologii testované jednotky
- jednoduší generování testů pro složité obvody a systémy, hlavně pro mikroprocesorové systémy

nevýhody

- dosáhnutí úplného či dostatečného pokrytí skutečných defektů je problém

### 2.1.3 Vestavěné diagnostické prostředky

Všechny dále popisované obvody se řadí do kategorie vestavěných diagnostických prostředků. Jejich cílem je provádění vnitřní diagnostiky pomocí zkoušeče přímo zabudovaného do testované jednotky, kterou budeme nazývat CUT (Core Under Test) – viz. Obrázek 2.3.




Obrázek 2.3

Lze je snadno použít pro autonomní testy bez nutnosti použití vnějšího zkoušeče. Výhodou je jednoduchá komunikace s okolím na úrovni jednoduchých signálů, či možnost provést test kdykoliv bez nutnosti připojování testeru, naopak nevýhodou malé využití technického vybavení použitého na vestavěnou diagnostiku a nutnost realizace v každé testované jednotce zvlášť.

My se zde budeme věnovat vestavěným zkoušečům realizovaným pro metodu komprese diagnostických dat a příznakovou analýzu. Jedná se moderní metodu realizace bez potřeby pomoci vnějších zkoušečů. Umožňuje miniaturizaci zkoušeče a její přímé umístění do jednotlivých stavebních modulů, především čipů. Tyto diagnostické prostředky se obvykle označují jako BIST (Bult-In Self-Test).

Jednotlivé typy vestavěného testování se od sebe liší v mnoha vlastnostech.Zde bych uvedl několik těch nejdůležitějších a nejvýraznějších hledisek, které je třeba brát v úvahu při jejich výběru.

- Prostorové nároky testu – velikost prostředků potřebných pro testování
- Časové nároky testu – doba potřebné pro test
- Kvalita testu – diagnostické pokrytí poruch daným testem

### 2.1.4 Autonomní testy generované v reálném čase

Generováním testu v reálném čase lze dosáhnout významné úspory technického vybavení věnovaného diagnostickým prostředkům. Místo testu samého se v testované jednotce uloží pouze předpis, podle něhož má být test generován.

Toho se využívá pro jednotky s pravidelnou strukturou, například paměti nebo strukturovaně navržené sekvenční obvody. Jeden ze způsobů, jak v reálném čase generovat test a komprimovat odezvu na něj, je obvodem přímo vestavěným do struktury testované jednotky. Nejčastější metodou je použití LFSR(Linear Feedback Shift Registers), neboli posuvného registru s lineární zpětnou vazbou.Tyto metody v současné době procházejí obdobím bouřlivého vývoje , proto lze očekávat jejich uplatnění ve stále větší míře.

## 2.2 Diagnostické prostředky

### 2.2.1 Vyhodnocování odezvy

Vyhodnocovač odezvy – ORA (Output Response Analyser) má zaškol zaznamenávat a kontrolovat odezvu na výstupech testované jednotky, která je reakcí na daný testovací vzorek na jejích vstupech. Tato odezva je nějakým způsobem přijímána a zpracovávána v průběhu testu a komprimována do konečné podoby, které je na konci testu porovnána s příznakem uloženým v paměti. Shoda s tímto očekávaným příznakem pak značí úspěšně provedený test, naopak rozdílná hodnota značí chybu.

Základním stavebním prvkem vyhodnocovače odezvy je příznakový analyzátor – SA (Signature Analyzer). Ten může být realizován mnoha způsoby, budu se však zde zabývat způsoben realizace pomocí LFSR, neboť ten používám v této práci.

Jelikož odezva testované jednotky na testovací vektory bývá obvykle velmi rozsáhlá (v závislosti na velikosti testovaného obvodu a délce testu), je důležité, aby vyhodnocovač odezvy dále plnil funkci komprese dat. Existuje více diagnostických metod, jejich společným cílem je snaha redukovat objem informace, kterou je potřeba přenášet, zaznamenávat popřípadě archivovat, na nejmenší možnou míru, při zachování podstatné části informačního obsahu potřebného pro diagnostiku. Důležitým požadavkem na kompresní metody je především jejich snadná realizovatelnost a malé nároky na technické vybavení.

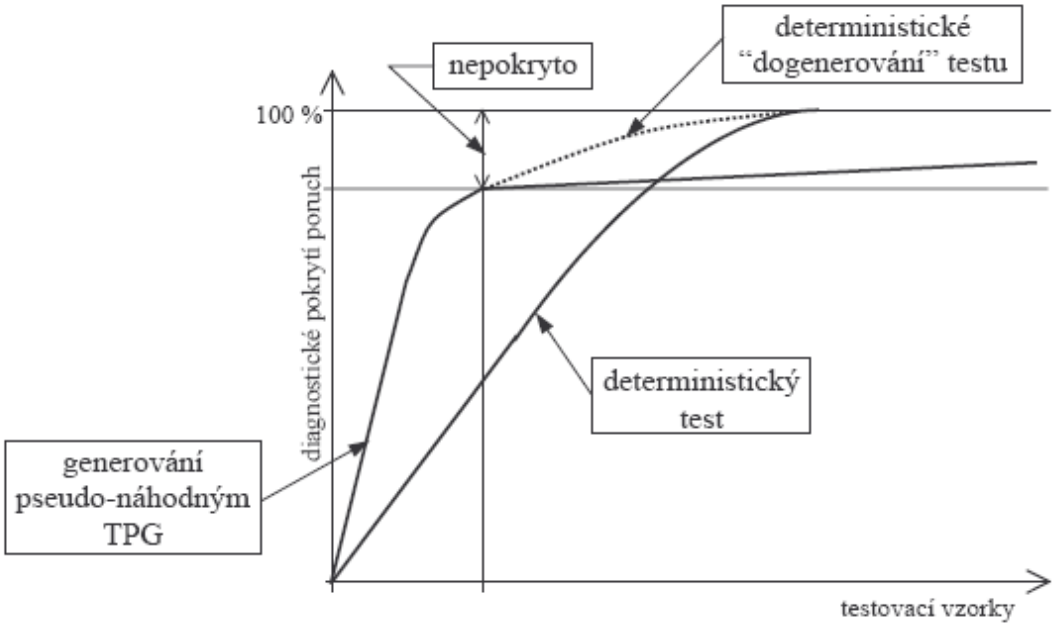
Mezi kompresními metodami jsou za velmi významné považovány metody pseudonáhodných transformací. Lze je souhrnně charakterizovat tak, že získaná odezva je nějakým jednoduchým předpisem transformována na krátké několika bitové slovo – příznak. Nejvýznamnější kategorie těchto metod jsou založeny na čítání událostí a dělení generujícím polynomem. Ta je v této práci realizována pomoci posuvného vícevstupového LFSR. Při takové kompresi dat dochází k určité ztrátě informace, proto může nastat situace, kdy pro se pro chybnou odezvu vytvoří stejný příznak jako pro bezchybnou.Takové poruchy se pak stávají nedetekovatelnými. Tato pravděpodobnost je však velice malá.

### 2.2.2 Pseudonáhodné generátory testovací posloupnosti

Pseudonáhodné generátory testovacích vzorku (TPG) se vyznačují svojí jednoduchou implementací a krátkým časem na vygenerování testu. Avšak poruchy, které lze testem vygenerovaným tímto TPG odhalit, patří pouze do kategorie lehko-detekovatelných poruch.

Lehká implementace tak bývá zpravidla vyvážena horším diagnostickým pokrytím poruch takového testu.

Pseudonáhodné TPG generují deterministické bitové kódy, které jsou jako testovací vzorky přiváděny na vstupy testovaného obvodu.

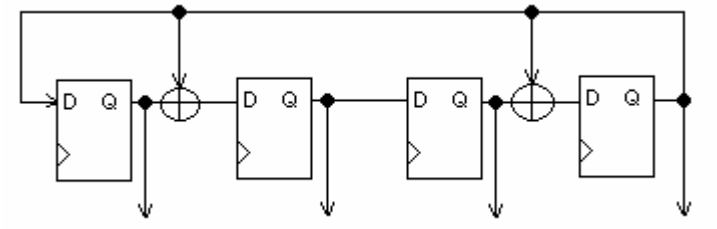



Obrázek 2.4 Pokrytí pseudonáhodným TPG

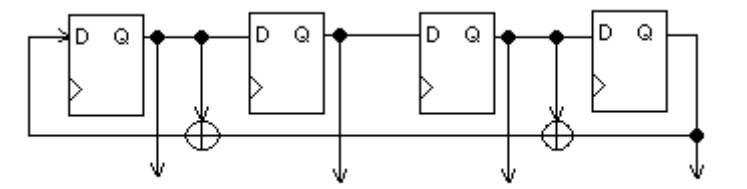
### 2.2.3 Linear Feedback Shift Registers (LFSR)

#### 2.2.3.1 Zapojení LFSR

Tyto lineární zpětnovazebné registry (dále jen LFSR) se nejčastěji používají jako zdroje testovací posloupnosti pro prostředky vestavěné diagnostiky. Důvodem je že jsou efektivnější a rychlejší než klasické binární čítače a také vyžadují méně kombinační logiky na jedno D hradlo, ačkoliv posloupnost jejich stavů je jiná a o jeden krok kratší. Máme dva základní typy implementace LFSR a to s vnitřním nebo vnějším zapojením zpětných vazeb – viz. Obrázek 2.5




(a) vnitřní zapojení zpětných vazeb



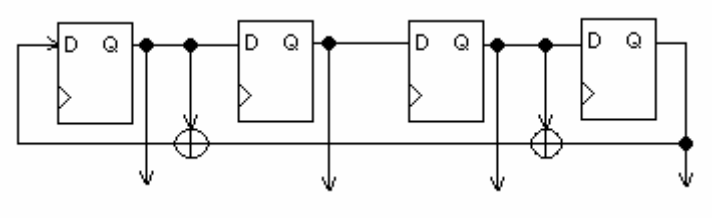

(b) vnější zapojení zpětných vazeb

Obrázek 2.5 Zapojení zpětné vazby LFSR

Obrázek 2.5b nejlépe ilustruje název obvodu: zpětná vazba je lineárně zapojena z jednotlivých členů (registrů) pomocí hradel XOR. Zde je provedeno zapojení podle primitivního polynomu $x^4 + x^3 + x^1 + 1$. Vnitřní i vnější zapojení LFSR je ekvivalentní a vyžaduje totožné množství logiky a registrů. Ačkoliv u vnějšího zapojení na obrázku 2.2b jsou v nejhorším případě na nejdelší cesta mezi výstupem posledního registru a vstupem do prvního dvě hradla XOR. Zatímco u zapojení s vnitřní zpětnou vazbou je na této nejdelší cestě maximálně jedno. Proto zapojení s vnitřní zpětnou vazbou přináší možnosti implementací s vyšší operační frekvencí pro náročné aplikace. Výhodou vnější zpětné vazby je větší jednotnost struktury logiky kolem registrů, což může být v některých aplikacích výhodou.

### 2.2.3.2 Primitivní polynomy

Posloupnost stavů, jež bude LFSR generovat je závislá především na primitivním polynomu, podle něhož je zapojena zpětná vazba – viz Obrázek 2.6 a 2.7.

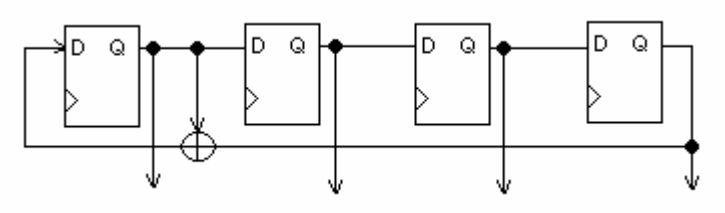



(a) vnější zpětná vazba $P(x)= x^4 + x^3 + x + 1$

Obvod bude procházet těmito cykly:

1. 0000->
2. 1111-> 1010-> 0101->
3. 0001-> 1101-> 1011-> 1000-> 0100-> 0010->
4. 1100-> 0110-> 0011->
5. 1001->
6. 1110-> 0001->

Obrázek 2.6




(b) vnější zpětná vazba $P(x)= x^4 + x + 1$

Obvod bude procházet těmito cykly

1. 0000->
2. 1111->1011-> 1011-> 1000-> 0100-> 0010-> 0001-> 1100-> 0110-> 0011-> 1101-> 1010-> 0101-> 1110-> 0111->

Obrázek 2.7

Z těchto obrázků je názorně vidět, že při zapojení vazeb podle primitivního polynomu $x^4 + x^3 + x + 1$ pro 4 registrový LFSR je maximální délka posloupnosti pouze 6, zatímco zapojení dle primitivního polynomu $x^4 + x + 1$ projde všechny stavy kromě nulového v jednom jediném cyklu.

Pro nás je pochopitelně nejzajímavější možnost, kdy LFSR projde nejdelší možnou posloupností stavů, což odpovídá $2^n$-1 stavům. Chybí zde nulový stav, protože z něho by se LFSR z principu funkce nikdy nedostal, a stejně tak pokud začne sekvenci stavů z jakéhokoliv nenulového stavu nemůže se do něho nikdy dostat. Takový stav je však pro účely testování beztak většinou nezajímavý. Znamená to však že do něho musí být na počátku testu vložen nenulový obsah. V tabulce 2.1 je přehled minimálních nenulových koeficientů primitivních polynomů pro jednotlivé počty registrů (2-16), po jejichž zapojení LFSR projde nejdelší možnou posloupnost stavů.

| Stupeň (počet registrů) | Primitivní polynom |
|---|---|
| 2, 3, 4, 6, 7, 15 | $x^n + x + 1$ |
| 5, 11 | $x^n + x^2 + 1$ |
| 8 | $x^n + x^6 + x^5 + x + 1$ |
| 9 | $x^n + x^4 + 1$ |
| 10 | $x^n + x^3 + 1$ |
| 13 | $x^n + x^4 + x^3 + x + 1$ |
| 14, 16 | $x^n + x^5 + x^4 + x + 1$ |

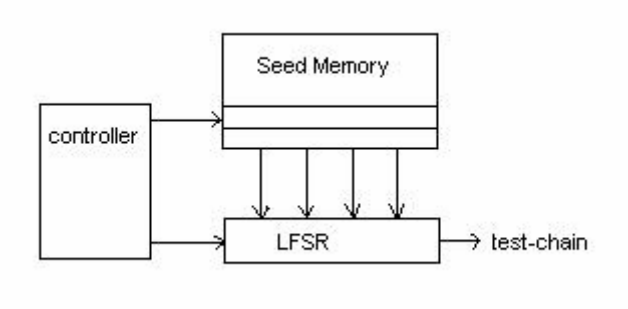
Tabulka 2.1 Primitivní polynomy

Jak můžete vidět z tabulky, minimální počet XOR hradel nutných k implementaci těchto primitivních polynomů se pohybuje mezi jedním až třemi.

### 2.2.4 Technika reseeding

TPG implementované jako LFSR projde pro daný primitivní polynom všechny stavy během $2^n$-1 taktů (z výjimkou nulového). Toto nám sice garantuje takřka úplné pokrytí, nicméně v praxi by to znamenalo pro daný obvod LFSR délku rovnou počtu délce testovaných vektorů, což by znamenalo dlouhé zpoždění na zpětné vazbě. Dále by takový test byl neúnosně dlouhý. Navíc pro úplné pokrytí testu není potřeba procházet všechny stavy.

Technika reseedingu využívá toho, že během relativně krátkého běhu pseudonáhodného generátoru testovacích vzorků - PRPG (Pseudo Random Patern Generator), je pokryta většina poruch, především ty lehko detekovatelné a dále přepokládá, že pro ty ostatní známe vektor pro jejich pokrytí. Po ukončení pseudonáhodné fáze jsou deterministicky degenerovány vektory pro pokrytí dosud nepokrytých chyb. To je provedeno vnucením stavu LFSR a následnému provedení testu danou posloupností pro pokrytí zbývajících chyb. Implementace je provedena přidáním paměti pro uložení jednotlivých stavů (Seedů), které chceme během testu nahrát do LFSR a řadiče pro ovládání, viz obrázek 2.8.

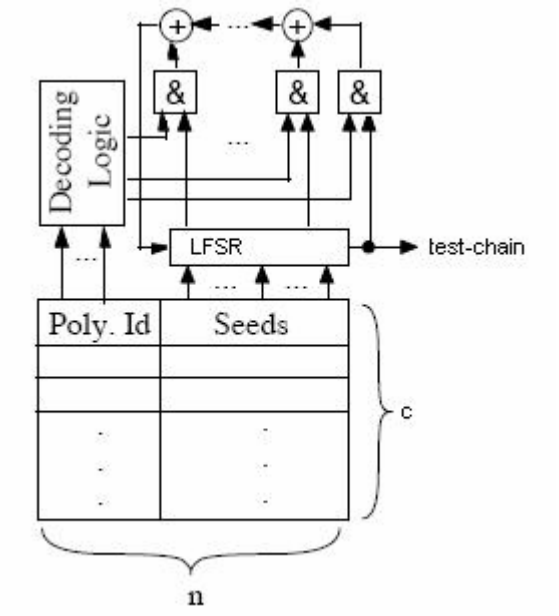



Obr. 2.8 Schéma pro techniku Reseeding

### 2.2.5 Multi-polynomialní LFSR

Pokud však vyžadujeme, aby námi používaný LFSR prošel některé konkrétní stavy, tj. vygeneroval určitou posloupnost stavů/testovacích vektorů, nemusí být primitivní polynom procházející všechny stavy ten nejlepší, neboť při takovém zapojení by sice LFSR všemi stavy prošel, nicméně za velmi dlouhou dobu. Dále pak, pokud požadujeme projít posloupnost konkrétních stavu ze stavu již známého (popř. LFSR vnuceného – viz. dále v části implementace MPLFSR), lze některými známými algoritmy pro takovou posloupnost spočítat potřebný polynom, který tuto posloupnost vygeneruje. (pozn. Tato úloha také v některých případech nemusí mít řešení).

Někdy pro požadujeme, aby bylo možné v TPG realizovaném pomocí LFSR měnit zpětnou vazbu tohoto LFSR přímo během prováděného testu. Např. po vnucení stavu v metodě reseedingu (viz dále v popisu implementace MPLFSR). Toto může být realizováno pomocí přidání dekodéru pro zpětnou vazbu, jako je to naznačeno na obrázku 2.9.




Obr. 2.9 Zapojení MPLFSR s dekodérem zpětné vazby

# 3. Analýza a postup řešení práce

V této části nastíním postup při řešení diplomové práce, výběr technik a metod implementace i problémy a řešení s nimiž jsem se během práce setkal. Rozeberu podrobně algoritmus použitý pro techniku reseedingu, který používám ve vytvořených programech a popíši do detailu strukturu generovaných VHDL komponent a jejich funkci.

## 3.1 Rozbor a výběr možných řešení

Jak napovídá zadání, hlavním pilířem práce bude implementace BISTu technikou reseedingu. Primárním cílem tak bylo vytvořit program, který načte obvod zadaný pomocí netlistu a vygeneruje VHDL kód tohoto obvodu a k němu připojenému testeru. Pomocí algoritmu, který rozeberu v další kapitole, se vygenerují testovací vektory – seedy, které se budou načítat z paměti do LFSR (viz. technika reseeding) k pokrytí ještě nepokrytých poruch. Tyto vektory budou uloženy v paměti testeru a za pomoci řadiče obsluhujícího běh testu postupně načítány k degenerování chyb, podle parametrů zadaných uživatelem. Program obsahuje přepínač do režimu manuálního zadání seedu uživatelem ze zdrojového souboru, takže tento algoritmus lze obejít a paměť testeru nakonfigurovat manuálně.

Existují dvě základní metody provádění testu vestavěnou diagnostikou: test-per-scan a test-per-clock. Tyto obě metody se od sebe výrazně liší ve velikosti technických prostředku pro realizaci a délce provádění testu díky odlišnému způsobu aplikování testovacích vzorku na testovaný obvod. Uživatel si může pomocí přepínače vybrat jakou metodou chce vygenerovat tester. V testování jsem se výrazně zaměřil právě na tento rozdíl obou metod a v kapitole testování jsou prezentovány výsledky měření a srovnání obou metod ve zkoumaných parametrech.

Dále je možné programu zadat jako parametr nula reseedů, tj. dojde k vygenerování pouze jednoduchého TPG implementovaného LFSR s jednou zpětnou vazbou a bez deterministického degenerování testu pomocí reseedingu. Toto řešení je úsporné o paměť obsahující seedy a jejich řízení, nicméně očekávané pokrytí takovým testem je čistě pseudo-náhodné a výrazně nižší. Více v sekci testování, kde je vše porovnáno.

Dalším způsobem jak zlepšit pokrytí poruch testem je přidání logiky pro dynamickou změnu zpětné vazby generujícího LFSR při zadání nového seedu tak, aby TPG v následujích stavech prošel takové stavy, kterými vygeneruje testovací posloupnosti, jež v dalších taktech pokryjí další poruchy, nejen pouze ty, které jsou pokryty vloženým seedem. Po resedu jsou totiž následující stavy LFSR určeny právě zpětnou vazbou a jejím nakonfigurováním dle vhodného polynomu, bychom tohoto mohli dosáhnout. Bohužel však algoritmy pro nalezení takového polynomu jsou značně náročné na naprogramování a vyžadují složité matematické počty a úpravy, proto jsme se z vedoucím práce po konzultaci rozhodli upustit od implementace tohoto algoritmu. Nicméně tuto techniku jsem naimplementoval, avšak bez algoritmu počítajícím reseedy a k nim zpětné vazby, takže program pro jejich generování nabízí pouze možnost manuálního zadání těchto parametrů uživatelem a zkoumání a implementování tohoto algoritmu ponechal na další práci.

Konečným výsledkem tak jsou 2 programy. Jeden pro tvorbu VHDL kódů pro techniku reseeding oběma zmíněnými způsoby s možností vygenerování seedů algoritmem nebo zadáním manuálně a druhý pro generování VHDL kódu pro

MPLFSR bez možností vygenerování seedů pomocí algoritmu, pouze s manuálním zadáním.

## 3.2 Algoritmus generování seedů pro reseeding

Cílem tohoto algoritmu je pro daný obvod nejprve odsimulovat pseudo-náhodnou fázi generování testovacích vektorů pomocí TPG implementovaného LFSR z pevnou zpětnou vazbou po určitý zadaný počet taktů. Poté pro zadaný počet reseedů opakovat sekvenci, která pro dané vygenerované testovací vektory zjistí pokrytí poruch a najde poruchy nepokryté. K těmto poruchám najde příslušné vektory pokrývající tyto poruchy a vybere z nich ten nejperspektivnější. Použje ho pro reseeding, odsimuluje následující reseedu a případné po něm následující takty a přidá nově generované vektory do sekvence těch již vygenerovaných pro další opakování algoritmu.

Posledním krokem pak je srovnání vytvořeného příznaku v příznakovém analyzátoru s očekávaným příznakem v paměti a vyhodnocení testu.

Pseudokód vytvořeného algoritmu pro techniku reseeding:

1. Simuluj počáteční pseudonáhodné generování vektorů
2. For(i; i<počet reseedů; i++)
   a. Najdi dosud nepokryté poruchy
   b. Najdi k nim vektory které je pokrývají
   c. Vyber z nich ten nejlepší
   d. Odsimuluj reseed a případné následující takty
   e. Přidej nové vektory
3. Srovnej příznak s očekávanou hodnotou
4. Vyhodnoť test

Některé tyto kroky jsou náročné a jsou obstarány programy proto specializovanými a obsaženými v adresáři s hlavním programem.+

## 3.3 Programy Bistgen

Program Bistgenr.exe slouží ke generování VHDL kódu obvodu s připojeným BISTem k netlistem zadanému obvodu. BIST používá techniku reseedingu, popřípadě samotného LFSR, při zadání nuly reseedů. Pomocí přepínačů/parametrů programu lze zvolit metodu testování test-per-clock nebo test-per-scan a také zvolit manuální zadání seedů namísto provedení implementovaného algoritmu pro jejich výpočet.

Program Bistgenr_mp.exe také slouží ke generování VHDL kódu obvodu s připojeným BISTem k netlistem zadanému obvodu. Ovšem používá techniku multi-polynomiálního LFSR s reseedingem. Pomocí přepínačů/parametrů programu lze zvolit metodu testování test-per-clock nebo test-per-scan. Jak již uvedl dříve, tento program umožňuje pouze manuální zadání seedů a k nim polynomů ze souboru.

Programy jsou napsány v programovacím jazyce C a k bezproblémové funkčnosti pod operačním systémem Windows nepotřebují žádné další programy či prostředí.

### 3.3.1 Načítání parametrů

Hlavní program nejprve načte vstupní parametry ze souboru *parameters.txt*, který musí být obsažen v adresáři s hlavním programem a v předepsaném formátu obsahuje všechny proměnné parametry, které zadává programu uživatel.

Jsou to:

- *délka LFSR (počet bitů)* - počet bitu LFSR fungujícího jako TPG
- *polynom pro zpětnou vazbu LFSR* – primitivní polynom pro zapojení zpětné vazby LFSR fungujícího jako TPG, zadává se binárně, viz. manuál programu
- *počet počátečních taktů LFSR* – počet taktů, po které poběží LFSR zpočátku v pseudo-náhodné fázi generování testovacích vektorů, než dojde k prvnímu reseedu
- *počet reseedů* – počet, kolikrát dojde k reseedingu LFSR fungujícího jako TPG
- *počet taktu po kazdem reseedu (0 - bude aplikován pouze seed)* – počet taktů, po které poběží test po každém novém nahraní seedu do LFSR, pokud je nulový dojde pouze k aplikování daného seed, jako testovacího vektoru po jeden tak testu
- *příznak pro srovnání a vyhodnoceni testu (8 bitu*) – vektor, jenž bude uložen do paměti a na konci testu srovnán s obsahem kompaktoru odezvy pro zjištění výsledku testu
- *reseedy (načteny pokud je aktivní přepínač –m)* – seedy, které budou použity, pokud si uživatel zvolí jejich manuální zadání

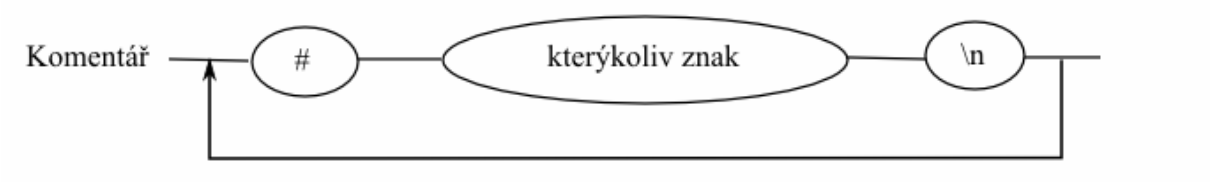
Tyto všechny parametry mohou být libovolně obměňovány dle potřeby uživatele programu, v jejich zdrojovém souboru se pouze požaduje jejich správné pořadí a uvození dvojtečkou.

Další potřebné parametry pro vygenerování BISTu k danému obvodu, jako je počet vstupů a výstupů obvodu a počet jeho klopných obvodů, jsou získány při načtení zdrojového souboru ve formátu netlistu a jeho překladu do VHDL formátu. Po spuštění programu a úspěšném načtení ze souborů jsou tyto parametry pro kontrolu a informaci vytištěny na standartní výstup.

### 3.3.2 Převod netlistu do VHDL

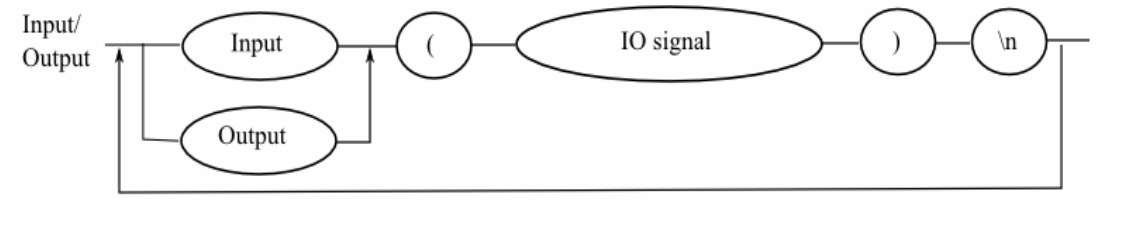
Následujícím krokem je překlad zdrojového souboru ve formátu netlistu do syntetizovatelné formy VHDL, pro správné načtení se požaduje, aby tento soubor měl stejný název jako je název obvodu (jež je zadáván jako parametr programu)a měl koncovku .bench (např. c432.bench). Překlad je proveden v několika krocích detekcí klíčových slov/znaku na řádku vstupního souboru, dle formátu netlist.

Znak # uvozuje komentář, takže na takto uvozené řádky není brán zřetel.
Syntaktický diagram:




Obr. 3.1 Syntaktický diagram pro komentář

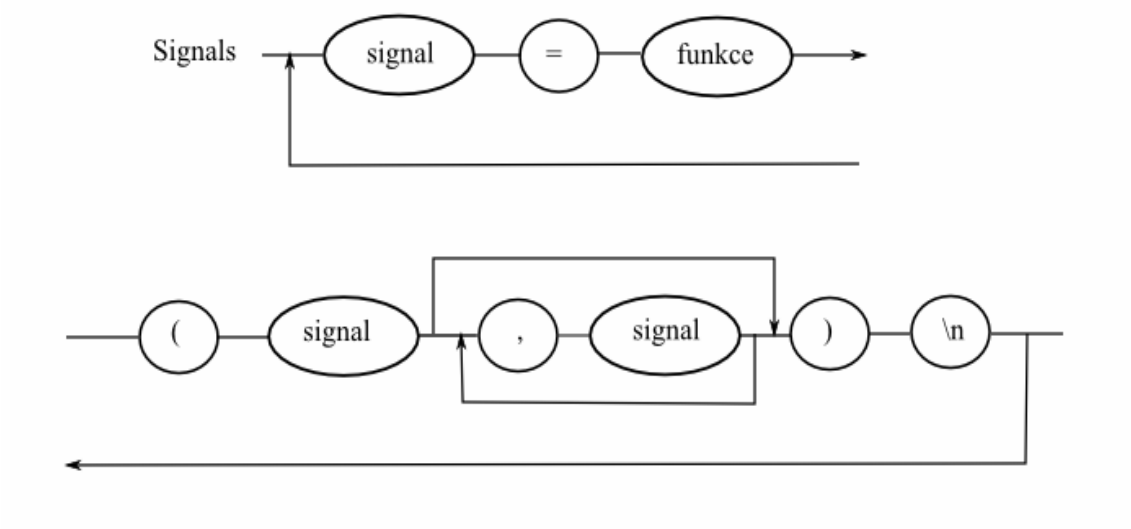
Input/Output uvozuje vstupy a výstupy. Ty jsou převedeny na vstupní a výstupní porty entity obvodu. Výstupní signály mají před název doplněn o. Vstupy a signály pak mají prefix s. To je z důvodu předpisu jazyka VHDL, který neumožňuje popis signálů pouze číslem, čemuž se tímto vyvarujeme. Abychom výstupní signály mohli deklarovat jako porty a nemuseli jsme složitě detekovat, které signálové přiřazení jsou přímo ty výstupní, označíme všechny signály prefixem s a tím vytvoříme signály i pro přiřazení přímo na výstupní porty. Tím se nám odlisují od výstupních portu právě pouze o tento prefix, takže nám stačí na konec přiřadit na signály výstupních portu tyto signály lišící se pouze prefixem.
Syntaktický diagram:




Obr. 3.2 Syntaktický diagram pro Input/Output

Po načtení a převedení všech vstupů a výstupů jsou převedeny signály a k nim příslušné funkce. Pro každý signál, který je výsledkem nějaké funkce, tj. pro každý signálový řádek v netlistu, se vygeneruje deklarace signálu ve VHDL pro takový signál a přiřazení jeho funkce se převede do popisu ve VHDL. Na závěr jsou výstupní signály přiřazeny na signály výstupních portů.

Syntaktický diagram:




Obr. 3.2 Syntaktický diagram pro Signály

Funkcím NAND a NOR musela být z důvodu syntetizovatelnosti věnována více pozornosti. Jejich forma musela být převedena na z prostého signálového spojení přes tuto funkci, jako je to u ostatních, na negaci funkce opačné. Neboť u více než dvou-vstupových signálových přiřazení hlásila syntéza error.

Příklad:

s1 <= s2 NAND s3 NAN s4; se převedlo na s1 <= NOT ( s2 AND s3 AND s4 );

Speciálním případem funkce je funkce DFF, která označuje D klopný obvod. Výskyt takové funkce označuje, že se jedné o obvod sekvenční. Pro obvody kombinační je část generovaného VHDL o něco jednoduší a menší, neboť není potřeba, převádět vnitřní klopné obvody testované jednotky na buňky potřebné pro testování dle metody (scan-par-clock/scan-per-chain). A také připojení k testeru je jednoduší.

Pro obvody sekvenční je nutné k VHDL obvodu vygenerovat také VHDL popis buňky, dle zvolené metody se liší, které jsou pak jako komponenty použity v obvodu namísto standardních D klopných obvodů pro potřeby testování. Ty pak všechny ve VHDL popisu naportovat na příslušné signály.

Jelikož algoritmem používané programy (Atlanta-M, Hope) pracují s obvody kombinačními, je pro jejich potřeby dále nutno sekvenční obvod převést/rozsekat na kombinační, tak aby mohl být použit pro jejich vstupy. Provede se to tak, že z obvodu vyjmeme všechny klopné obvody a signály jež do nich vedou označíme jako pseudo-výstupy – PPO (Pseudo-Primary Outputs) obvody a ty jež vedou z nich pak jako pseudo-vstupy – PPO (Pseudo-Primary Inputs) nyní kombinačního obvodu.

Pro tento účel program vytvoří kopii nelistového zdrojového souboru se stejným názvem a postfixem c, který pak je použit jako vstup zmíněných programů. Ta neobsahuje řádky pro funkce DFF, ale jsou nahrazeny každá jedním řádkem označujícím vstupy a jedním pro výstup těchto pseudo-vstupů a pseudo-výstupů, jimiž

jsme D klopný obvod nahradili. Pro testování to pak znamená, že počet vstupů a výstupů obvodu se zvýší právě o počet vnitřních kopných obvodů testované jednotky.

### 3.3.3 Generovaním BISTu

K provedení algoritmu uvedeného v kapitole 3.1 využívá hlavní program několik externích programů, které postupně v sekvencích spouští s vygenerovanými parametry. Jedná se o programy mnou vytvořené - lfsrbegin.exe, lfsr.exe, vhdlgen.exe a programy od jiných autorů – atalanta-M.exe, hope.exe.

Program vytvoří soubor s názvem obvodu a koncovkou .bat, do kterého vygeneruje tuto posloupnost příkazů na spuštění programů a tento soubor na konci svého běhu spustí. Jednotlivé spuštění těchto programu s parametry vycházejícími ze zadání a výpočtů, tak odpovídá některým krokům algoritmu, zvláště se jedná o ty, které se v cyklu opakují.

Nyní konkrétně k funkci těchto programů:

lfsrbegin.exe

Provádí první fázi pseudo-náhodného testování. Simuluje činnost LFSR podle zadaných parametrů (dálka, zpětné vazba) a do souboru *vektors.vec* generuje testovací vektory o délce rovné počtu vstupů testovaného obvodu (PI + PPI). Těchto vektorů bude tolik kolik je zadána délka počátečního běhu LFSR v pseudonáhodné fázi testu.

Příklad volání program:
lfsrbegin.exe s344 24 20

- parametry jsou: název obvodu, počet vstupů obvodu a počet požadovaných taktů, jež má být odsimulováno

lfsr.exe

Program načítá ze souboru *<název obvodu>.pat* (ten v předchozím kroku bývá vytvořen programem Atlanta-M.exe) jednotlivé vektory pro nepokryté poruchy a k nim vektory určující, jaké poruchy daný vektor pokrývá. Vybírá z této posloupnosti ten nejlepší, což znamená, že vybere první takový, který pokrývá nejvíce nepokrytých poruch.

Tento vektor určí, že bude použit pro reseeding a nahraje ho do souboru *reseeds.vec* odkud jsou na závěr tyto vektory generovány do VHDL kódu paměti seedů. Následně pak odsimuluje takt reseedu a případných dalších taktu po něm následujích dle předpisu generujícího LFSR a tyto vektory připíše do souboru *vektors.vec*.

Příklad volání programu:
lfsr.exe s344 24 5

- parametry jsou: název obvodu, počet vstupů obvodu a počet požadovaných taktů po daném reseedu, jež má být odsimulováno

vhdlgen.exe

Tento program se vždy volá na konec algoritmu a slouží k vygenerování komponent BISTu v jazyce VHDL. Má dvě verze, které se volají v závislosti na vybrané metodě testování, neboť testery se v obou verzích od sebe výrazně liší.

Příklad volání programu.
vhdlgen_r_si.exe s344 24 26 20 5 0

- parametry jsou: název obvodu, počet vstupů obvodu, počet výstupů obvodu, počet počátečních taktů LFSR, počet reseedů, počet taktů po reseedu

Hope.exe

V algoritmu tento program slouží k analýze testovacích vektoru a vygenerování seznamu nepokrytých poruch. Vstupem je vždy předchozími programy vytvořený nebo modifikovaný soubor *vektors.vec*, výstupem pak soubor *<název obvodu>.uf1* obsahující seznam nepokrytých chyb, který je pak dále zpracován programem Atlanta-M.exe. Dále je jeho výstupem do logfilu *<název obvodu>.lgc* a na standardní výstup info o získaném pokrytí poruch obvodu. Obsahuje informace o procentuálním pokrytí všech možných poruch obvodu a také seznam použitých vektoru a k nim odhalené poruchy a další užitečné informace.

Příklad volání programu:
hope -t vektors.vec -l s344.lgc -D -U s344.uf1 s344c.bench

- parametry jsou: zdrojový soubor s vektory, cílový logfile, přepínač do diagnostického módu, jež požadujeme, cílový soubor pro seznam nepokrytých poruch a soubor popisující testovaný obvod ve formátu netlistu

Atalanta-M.exe

Program je používán pro vygenerování testovacích vektorů pro nepokryté poruchy. Vstupem je pro něho seznam nepokrytých poruch *<název obvodu>.uf1* vytvořený programem Hope.exe. Výstupem pak soubor *<název obvodu>.pat* obsahující seznam vektorů a k nim poruchy jež tyto vektory pokrývají. Ten je dále zpracováván programem lfsr.exe.

Příklad volání programu:
atalanta-M -D 5 -t s344.pat -P s344.rep -f s344.uf1 -W 4 s344c.bench

- parametry jsou: přepínač do požadovaného režimu, cílový soubor pro testovací vektory, soubor s reportem, zdrojový soubor s nepokrytými poruchami a soubor popisující testovaný obvod ve formátu netlistu

Průběh volání těchto programů k provedení těchto algoritmů:

Jako první se vždy spouští program lfsrbegin.exe, k simulaci počáteční fázi pseudo-náhodného testování a následně se na vygenerovaný seznam vektoru pustí program Hope.exe. Pokud je zvolena nula reseedů, tj. pouze obyčejný LFSR bez reseedingu, obsahuje algoritmus pouze tato dvě volání plus koncové volání programu vhdlgen.exe pro generování VHDL BISTu.

Poté se v cyklu, jejichž počet se rovná počtu reseedu opakuje volání programů pro výpočet seedu. Ten začíná spuštěním programu Atlanta-M.exe pro vygenerování testovacích vektorů pro nepokryté poruchy, jež jsme získali pomocí Hopu. Z těchto vektorů se pomocí programu lfsr.exe vybere ten nejlepší a odsimuluje se jím reseed. Pro updatovaný soubor s testovacími vektory se opět provede zjištění pokrytí a nepokrytých poruch pomocí programu Hope.exe a smyčka se může znovu opakovat pro další reseed.

Jako poslední se pak spustí program vhdlgen.exe pro vygenerování VHDL komponent BISTU.

Příklad postupného volání programů, čili případný obsah souboru s koncovkou bat:

```
lfsrbegin.exe s344 24 20
hope -t vektors.vec -l s344.lgc -D -U s344.uf1 s344c.bench
atalanta-M -D 5 -t s344.pat -P s344.rep -f s344.uf1 -W 4 s344c.bench
lfsr.exe s344 24 0
hope -t vektors.vec -l s344.lgc -D -U s344.uf1 s344c.bench
atalanta-M -D 5 -t s344.pat -P s344.rep -f s344.uf1 -W 4 s344c.bench
lfsr.exe s344 24 0
hope -t vektors.vec -l s344.lgc -D -U s344.uf1 s344c.bench
vhdlgen_r_si.exe s344 24 26 20 5 0
```

Pokud si uživatel zvolí manuální zadání seedů ze souboru (přepínač -m), pak je celá sekce pro provádění algoritmu jejich výpočtu programem přeskočena a tyto zadané seedy jsou použity pro paměť seedů v BISTu.

Pro program bistgenr_mp.exe, tj pro implementaci BISTU technikou multi-polynomiálního LFSR s reseedingem, je toto zadání ze souboru jedinou možností. Tím pádem nejsou pro tento program potřeba žádné pomocné programy. Naopak navíc je zde implementován algoritmus pro rozšíření bitů seedů o bity určující zpětnou vazbu. Kde n bitů je potřeba pro dekódování $2^n$ zpětných vazeb.

Manuály popisující práci s programy, zadávání parametrů, jejich výstupy atd. jsou přiloženy v adresáři s daným programem. Mají stejný název a koncovku .man. Dále jsou přiloženy jako příloha této diplomové práce na jejím konci.

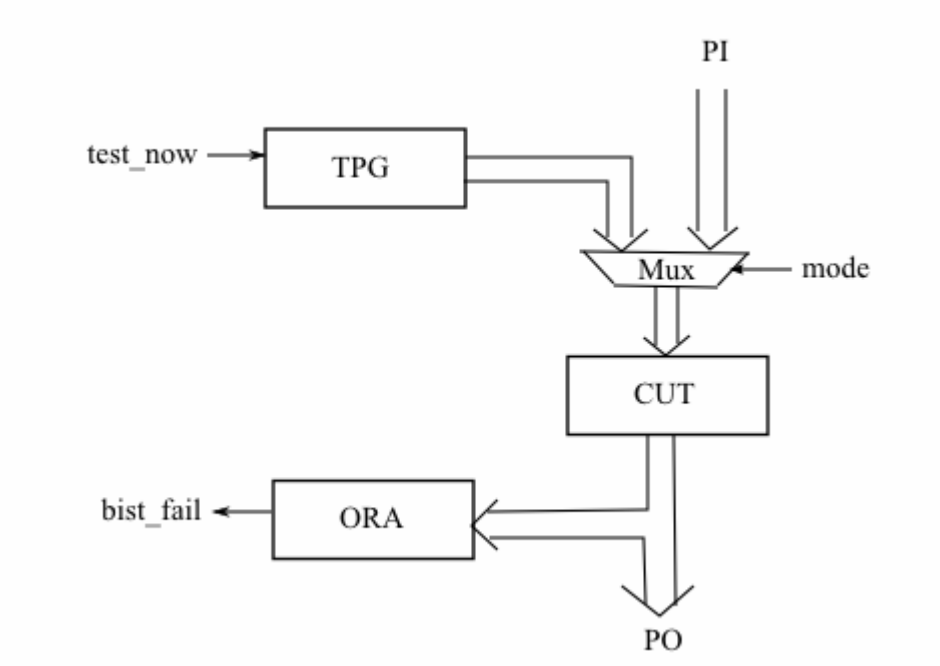
# 4. Popis implementovaných součástí

V této části dopodrobna rozeberu implementované součásti BISTů s reseedingem generovaných vytvořenými programy. A to pro obě dvě metody testování a v poslední části pak přiblížím a rozeberu implementaci multi-polynomiálního LFSR s reseedingem.

## 4.1 Metoda test-per-clock

Jedná se o metodu, která se snaží především o co nejrychlejší otestování během co nejméně taktů. Naopak slabina spočívá v možných větších nároků na plochu a především rozsáhlejšího propojení testeru s testovaným obvodem.

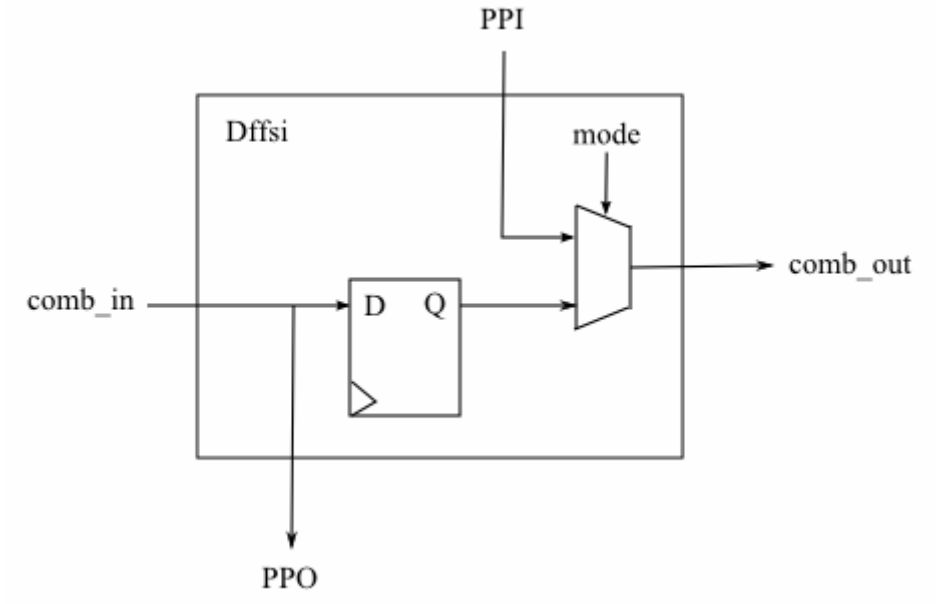
Základní schéma propojení CUT s testerem je na obrázku 4.1.




Obr. 4.1 Základní schéma propojení pro metodu scan-per-clock

Propojení testeru k testovanému obvodu je provedeno paralelně, což umožňuje provést test na každý hodinový signál po který je test vygenerován. Stejně tak je na výstupech obvodu paralelně na každý hodinový takt přijímána odezva obvodu na testovací vektory a je prováděna komprese této odezvy. To vyžaduje na straně testeru na vstupech i na výstupech počet buněk – klopných obvodů roven počtu vstupů, potažmo výstupů. Vstupy a výstupy jsou pak k těmto buňkám připojeny paralelně, takže na hodinový signál vstupuje testovací vektor uložený v těchto buňkách do obvodu a zároveň do výstupních buněk vstupuje odezva na tento test.

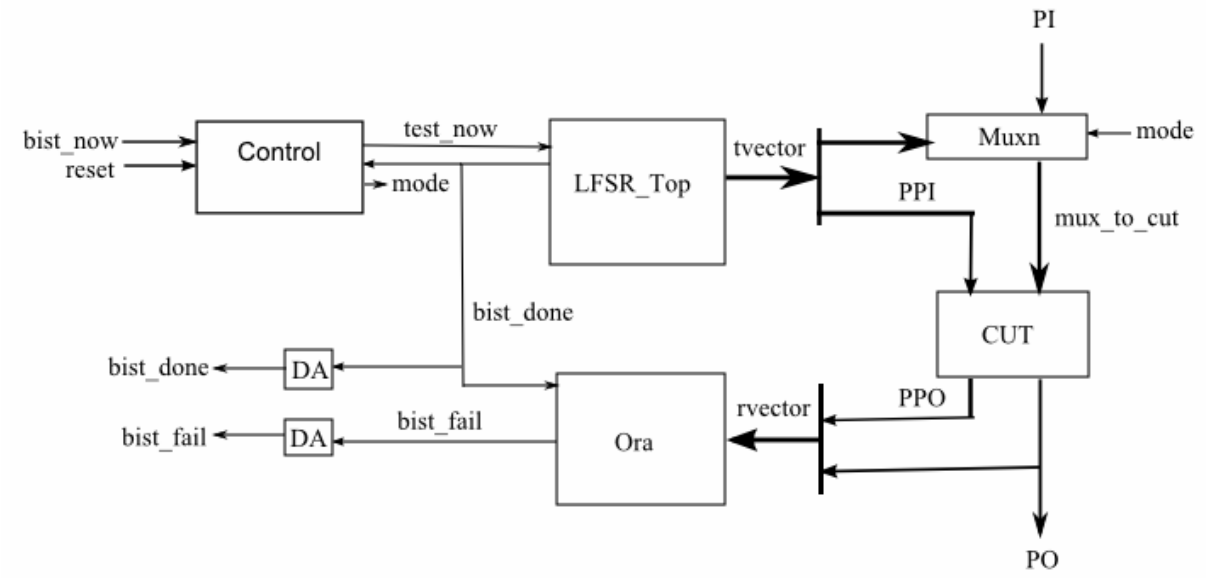
.




Obr 4.2 Schéma buňky Dffsi

Vnitřní klopné obvody v testované jednotce jsou převedeny na buňky Dffsi (na obrázku 4.2), umožňující po dobu testu převod sekvenčního obvodu na kombinační. To je provedeno tak, že výstupy z kombinační části obvodu jsou převedeny na pseudo-výstupy (PPO) z obvodu a vstupy z klopného obvodu zpět do kombinační části na pseudo-vstupy (PPI) obvodu. Mimo dobu testu pak tyto buňky normálně vykonávají svou funkci D klopných obvodů

Pro sekvenční obvody tak počet vstupů obvodu pro test naroste o počet vnitřních D klopných obvodu testované jednotky. TPG BISTu se skládá z LFSR (proměnné délky) a napojeného shift-registru pro případné nastavení délky do počtu vstupů (PI + PPI). ORA se skládá z MISRu, kterému stejně tak předchází shift-register pro případné nastavení délky výstupů (PO + PPO), komparátoru a paměti příznaku. Dalšími částmi BISTu je komponenta Control, jež obsahuje řadič testu a dále Muxn multiplexor, přepínajícími mezi primárními vstupy v pracovním módu a vstupy z testeru v módu testování. Schéma propojení celého obvodu je na obrázku 4.3.




Obrázek 4.3 Zapojení komponent obvodu metodou test-per-clock

Podrobnější popis komponent obvodu:

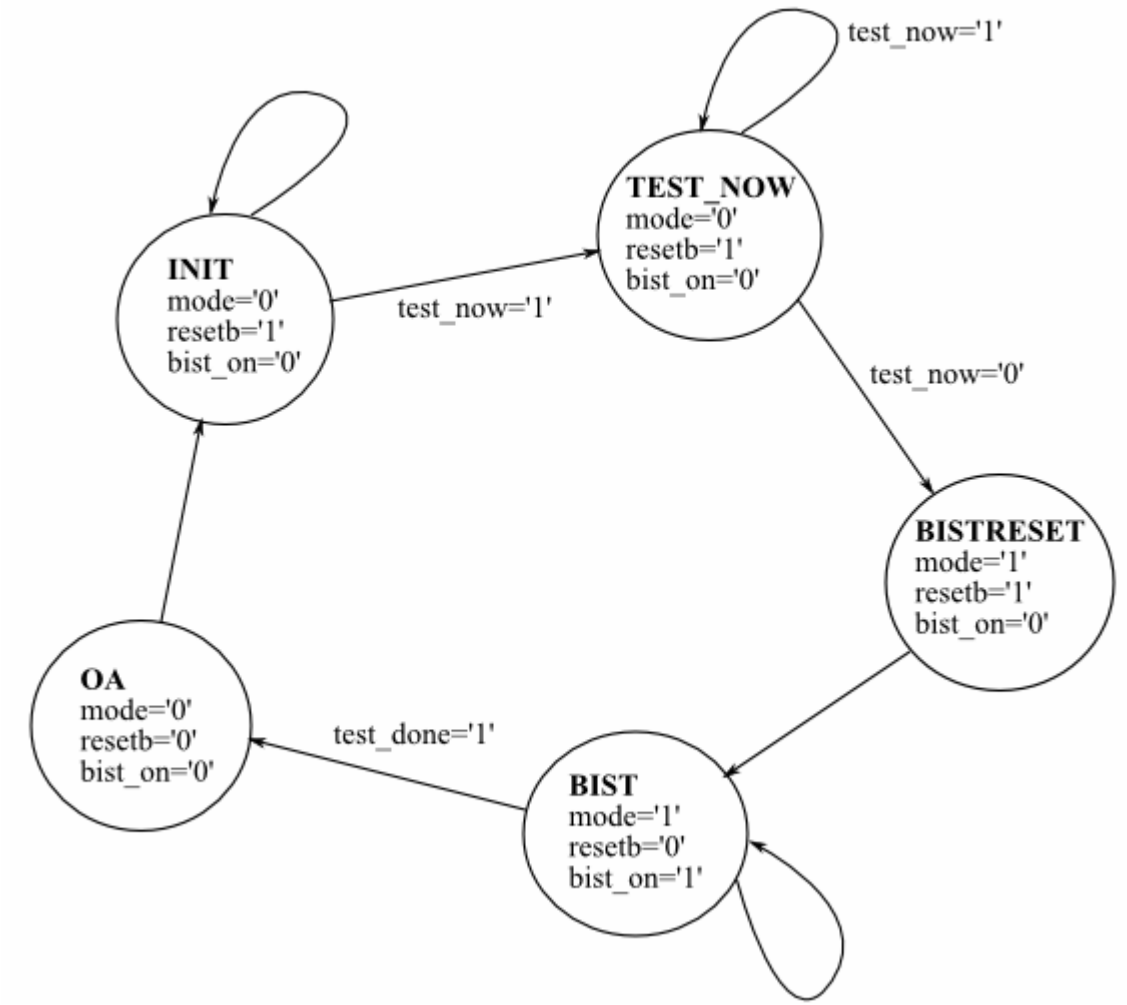
CUT
Testovaný obvod (Circuit Under Test).

Muxn
Multiplexor na primárních vstupech obvodu. V režimu testování přepne na vstupy z testeru.

DA_flop
D klopné obvody, pro zapamatování si výsledných signálu testu.

Control

Radič obvodu, má za úkol obsluhu testování. Graf přechodů řadiče je na obrázku 4.4. Testovaný obvod pracuje v normálním režimu a řadič čeká ve stavu Init dokud nepřijde signál test_now, který přepne do stavu Test_now.

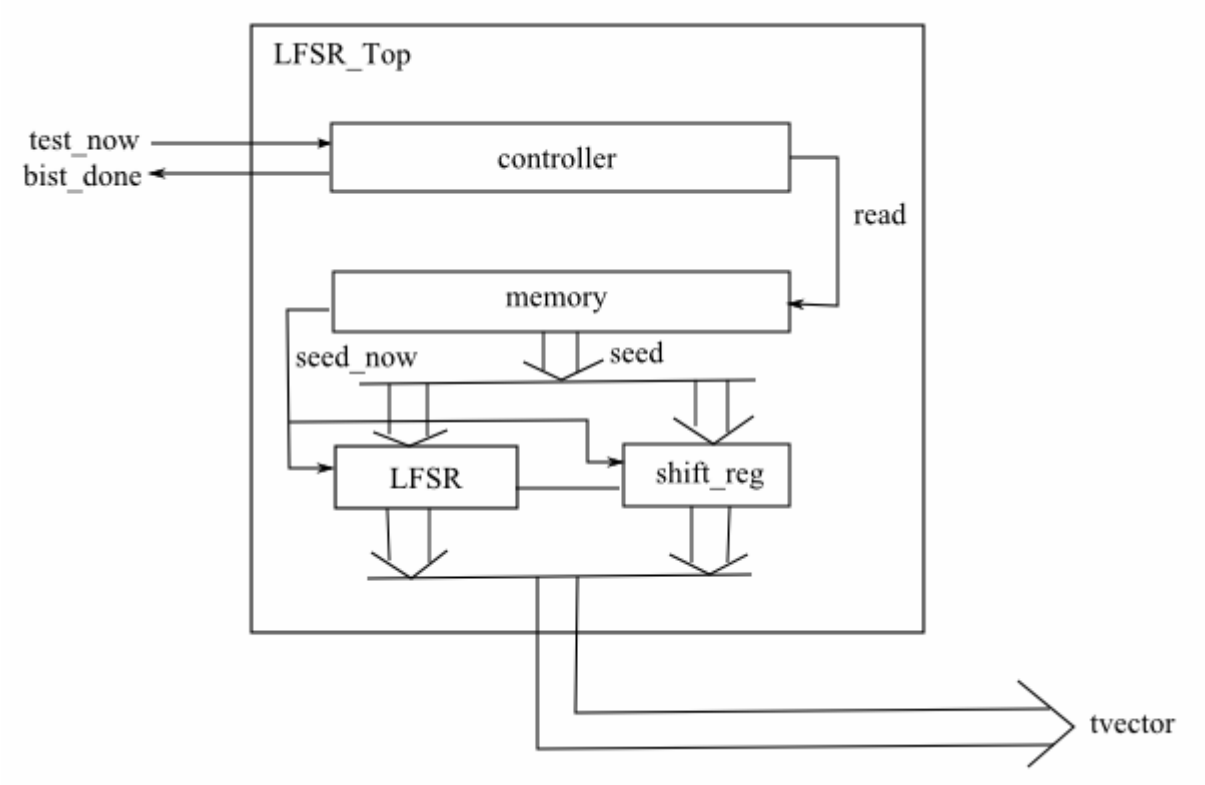



Obrázek 4.4 Graf přechodů řadiče pro metodu test-per-clock

V tom čeká na uvolnění tlačítka, potažmo signálu pro testování a poté přejde do stavu Bistreset, ve kterém resetuje BIST a všechny jeho součásti a na další hodinový signál započne testování – přechod do stavu Bist. Samotné řízení počtu taktu a reseedingu má na starosti řadič-čítač (komponenta Controller) uvnitř komponenty LFSR_Top (více o tom dále) , který na konci testu vyšle signál bist_done, který signalizuje konec testu a posune řadič do stavu Oa, který trvá jeden takt, potřebný pro srovnání příznaku v kompresoru odezvy s očekávanou hodnotou v paměti. Výsledkem testu je pak signál bist_fail (1 v případě testu bez odhalené poruchy).

LFSR_Top

Entita, jež v sobě obsahuje LFSR, představující TPG, nastavený Shift registrem. Dále paměť seedů a čítač/řadič, pro ovládání reseedingu. Schéma zapojení entity je na obrázku 4.5.




Obrázek 4.5 Zapojení entity LFSR_Top pro metodu scan-per-clock

LFSR a shift_reg

LFSR má délku zadánu parametry ze zdrojového souboru (volitelná). Je-li počet vstupů testovaného obvodu větší, je nastaven shift registrem, neboť pro metodu scan-per-clock je potřeba mít paralelně připojeny k testeru všechny vstupy testovaného obvodu. Jeho zpětná vazba je stejně tak zapojena dle zvolení uživatelem. O zapojení LFSR je více pojednáno v kapitole tomu věnované, viz. předchozí. Na vstupech do jednotlivých buněk LFSR a stejně tak shift registru je umístěn multiplexor, který na aktivní signál *seed_now* přepíná vstup z předchozí buňky (standardní režim generování testovacích vektoru) na vstup z paměti seedu. Ten je vždy nahrán v jednom taktu paralelně z paměti seedu.

Memory

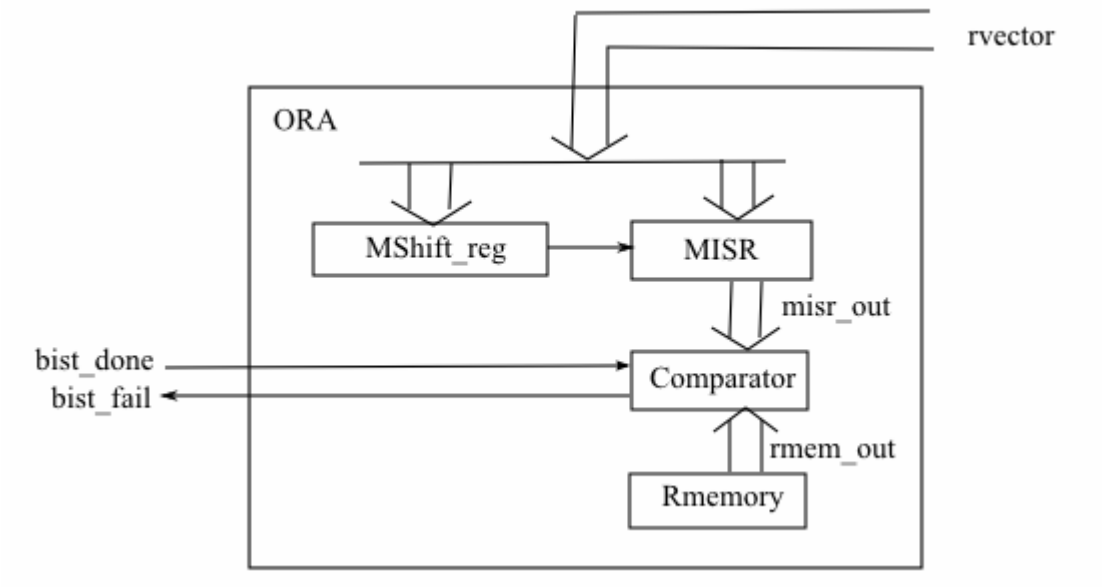
Paměť seedů. Na aktivní signál *read* pošle na sběrnici *seed* následující seed a zaktivní signál *seed_now*, takže seed je načten do LFSR a shift registru.

Controller

Tato entita obsahuje čítač pro čítaní počtu taktů testu. Čítání začíná na signál *test_now*. V požadovaných taktech vyšle signál pro načtení nového seedu z paměti. Na konec, po provedení požadované délky taktů, vyšle signál *test_done*, značící konec testu.

ORA

Neboli Output Response Analysator. Má za úkol přijímat na svých vstupech paralelně výstupy a komprimovat je v 8-mi bitovou odezvu. Z čehož pro metodu scan-per-clock vyplívá omezení testovaného obvodu na minimální počet výstupů 8. Toto však nepovažuji za příliš omezující, neboť se počítá s testováním obvodů s dostatečným počtem výstupů (PO) a vnitřních klopných obvodu (PPO). Dále pak po ukončení testu porovnává hodnotu v příznaku s očekávanou hodnotou v paměti. Schéma zapojení na obrázku 4.6. Skládá se z několika komponent.




Obrázek 4.6 Zapojení komponenty ORA pro metodu scan-per-clock

MISR a MShift_reg

Přijímají na svých vstupech paralelně výstupy obvodu (odezvy na testovací vektory) a komprimují je na 8-mi bitovou odezvu. Komprese je obstarávaná blokem MISR. Což je 8-mi bitový LFSR s primitivní zpětnou vazbou zapojený do série za shift registr. Na konci testu je na jeho paralelních výstupech konečný příznak testu.

Comparator

Na aktivní signál *bist_done* porovná hodnotu vystupující z MISRu s hodnotou v Rmemory a vyhodnocení vyšle na signál *bist_fail*, kdy 1 znamená, že test neodhalil žádnou poruchu.

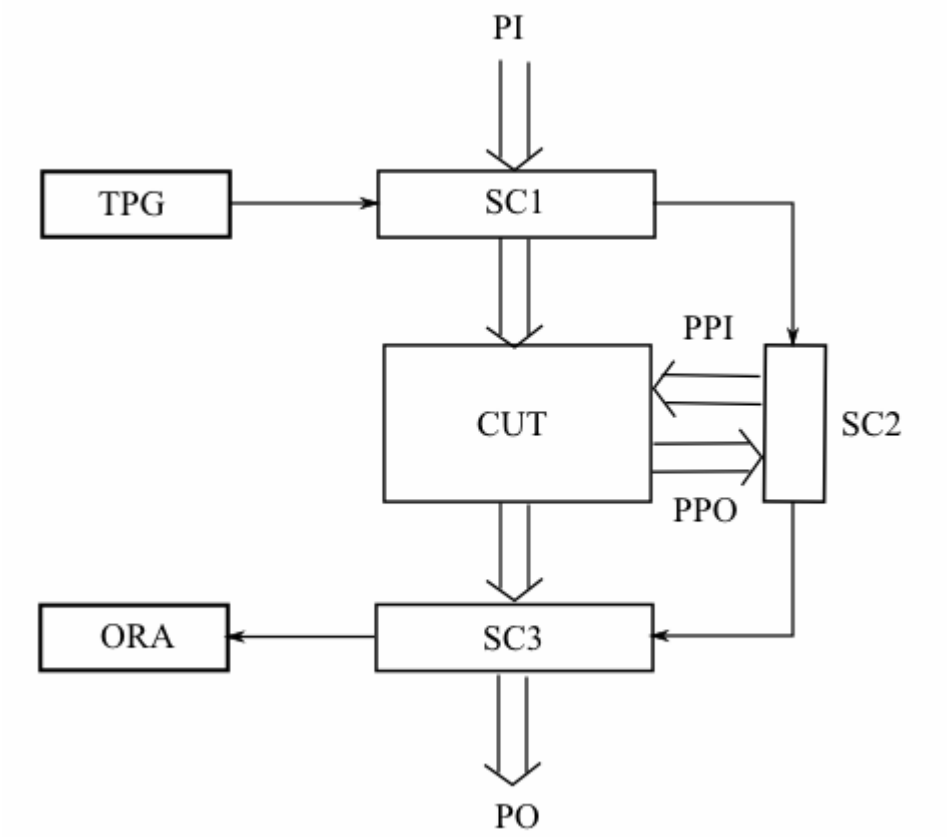
Rmemory

Paměť obsahující hodnotu (8b), kterou očekáváme na konci testu v kompresoru odezvy. Ta je zadávána uživatelem.

## 4.2 Metoda test-per-scan

Tato metoda používá pro propojení testeru a testované jednotky sériově propojené scan-chain buňky, kterými se “obalí“ obvod. Ty pak pracují v několika režimech. Při testování se nejprve testovací vektor sériově nasune do scan-chainu. Poté se paralelně provedou takty testu – vstup nasunutého vektoru do obvodu a zároveň se na výstupech obvodu do scan-chainu paralelně přijímá odezva. A jako poslední se výstupní scan-chain sériově vysune do kompresoru odezev, kde se tak vytvoří příznak.

Základní schéma propojení CUT s testerem je na obrázku 4.7.

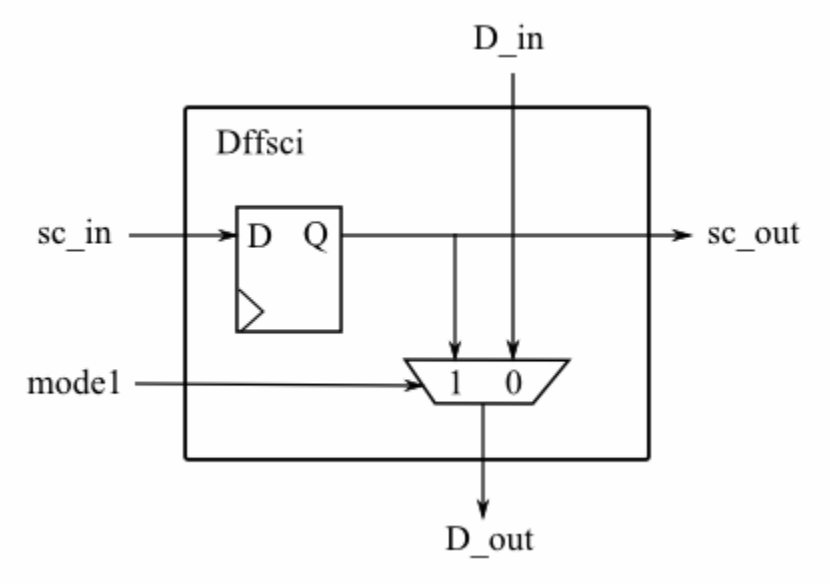



Obrázek 4.7 Základní propojení pro metodu test-per-chain

Jednotlivé buňky scan-chainu se od sebe liší pro tři různé jeho části. SC1 je vstupní scan-chain, SC2 pak odpovídá propojeným buňkám, jimiž jsme nahradili vnitřní klopné obvody tetované jednotky a SC3 je pak výstupní scan-chain.

Vstupní scan-chain buňka Dffsci

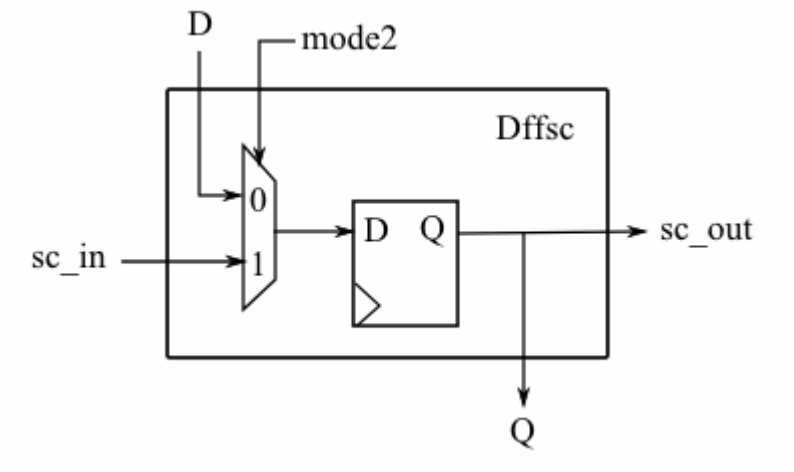
Tyto buňky zapojené sériově za sebou tvoří vstupní scan-chain. V normálním pracovním režimu obvodu pouze propojují primární vstupy na vstupy obvodu. V testovacím režimu buďto vysílají do obvodu obsah D klopných obvodů. Nebo po přepnutí multiplexoru slouží jako posuvný registr pro nasunutí testovacího vektoru. Schéma zapojení této buňky je na obrázku 4.8.




Obrázek 4.8 Schéma buňky Dffsci

Vnitřní scan-chain buňky Dffsc

Pokud je obvod sekvenční tak se těmito buňkami nahradí vnitřní D klopné obvody testované jednotky. A zapojí se sériově do scan-chainu. V normálním pracovním režimu obvodu se jejich funkce od běžných D klopných obvodu nijak neliší. Po přepnutí pak fungují jako sériový shift register. V tomto módu se do nich sériově nasouvá testovací vektor, nebo se z nich sériově vysouvá odezva obvodu. Schéma zapojení této buňky je na obrázku 4.9.

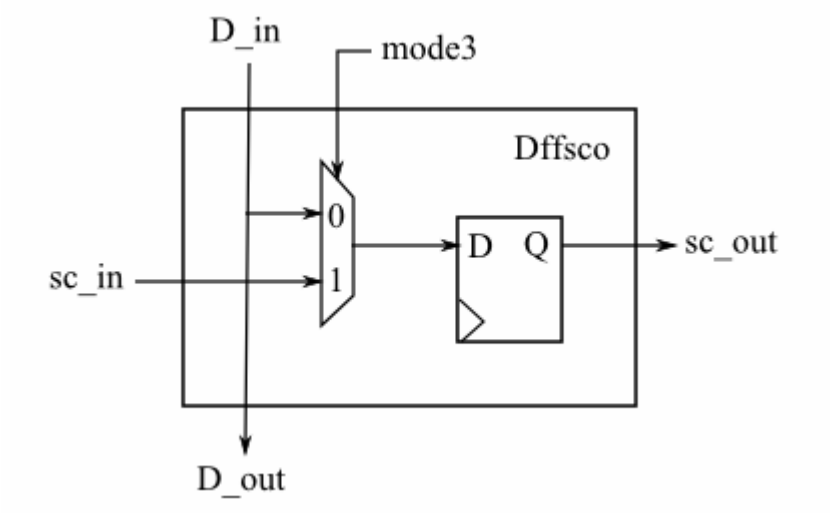



Obrázek 4.9 Schéma buňky Dffsc

Výstupní scan-chain buňky Dffsco

Tyto buňky zapojené sériově za sebou tvoří výstupní scan-chain. V normálním pracovním režimu obvodu pouze propojují výstupy obvodu na primární výstupy. Ve stejném módu pak taká fungují při testu jako přijímač odezvy, kdy vstup z kombinační části obvodu se zapíše do D klopného obvodu. V druhém režimu pak představují shift

registr pro sériové vysunutí přijaté odezvy pro kompresy na příznak. Schéma zapojení této buňky je na obrázku 4.10.




Obrázek 4.10 Schéma buňky Dffsco

Tyto tři scan chainy jsou sériově zapojeny za sebe, tak jak je to na obrázku 4.7. Pokud je obvod pouze kombinační a nemá tak žádné vnitřní D klopné obvody, tester neobsahuje vnitřní scan-chain a sériový výstup z vstupního scan-chainu je zapojen do sériového vstupu výstupního.

TPG BISTu se skládá z LFSR (proměnné délky) a napojeného shift-registru pro případné nastavení délky do počtu vstupů (PI + PPI), pro paralelní nahrání seedů. ORA se skládá z 8-mi bitového MISRu, komparátoru a paměti příznaku. Dalšími částmi BISTu je komponenta Control, jež obsahuje řadič testu a komponenta Counter, což je čítač pro obsluhu taktů LFSR a reseedingu. Schéma propojení celého obvodu je na obrázku 4.11.

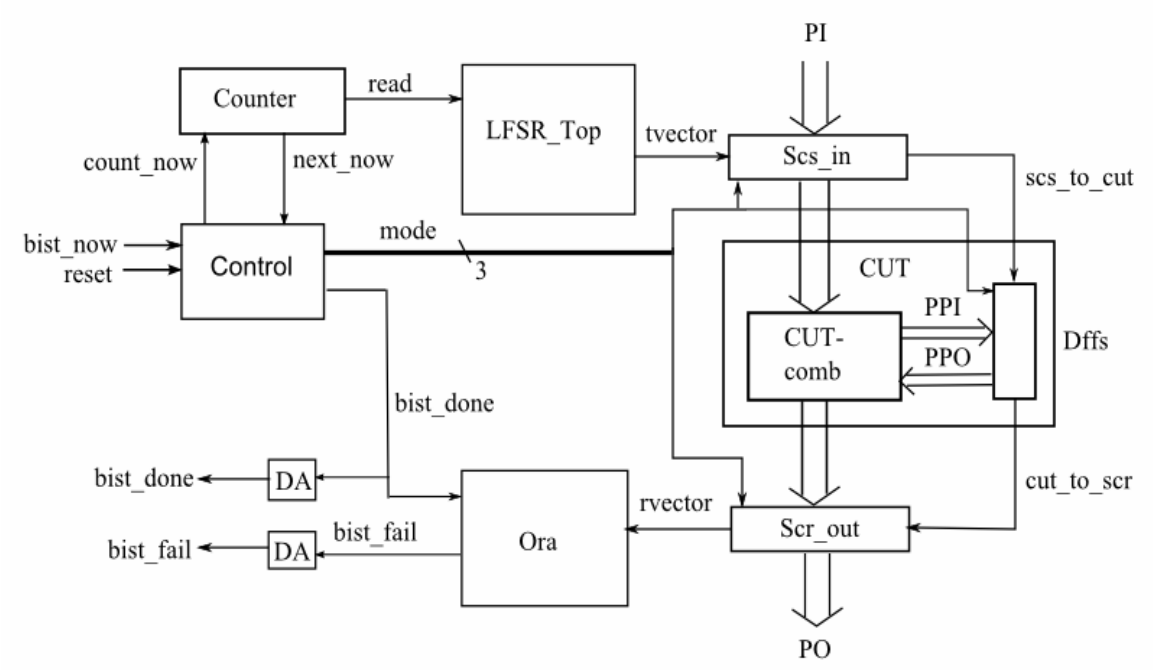
Podrobnější popis komponent obvodu:

CUT
Testovaný obvod (Circuit Under Test).

Scs_in
Vstupní scan-chain.

Scr_out
Výstupní scan-chain.

DA_flop
D klopné obvody, pro zapamatování si výsledných signálu testu.




Obrázek 4.11 Schéma obvodu pro metodu test-per-scan

Counter

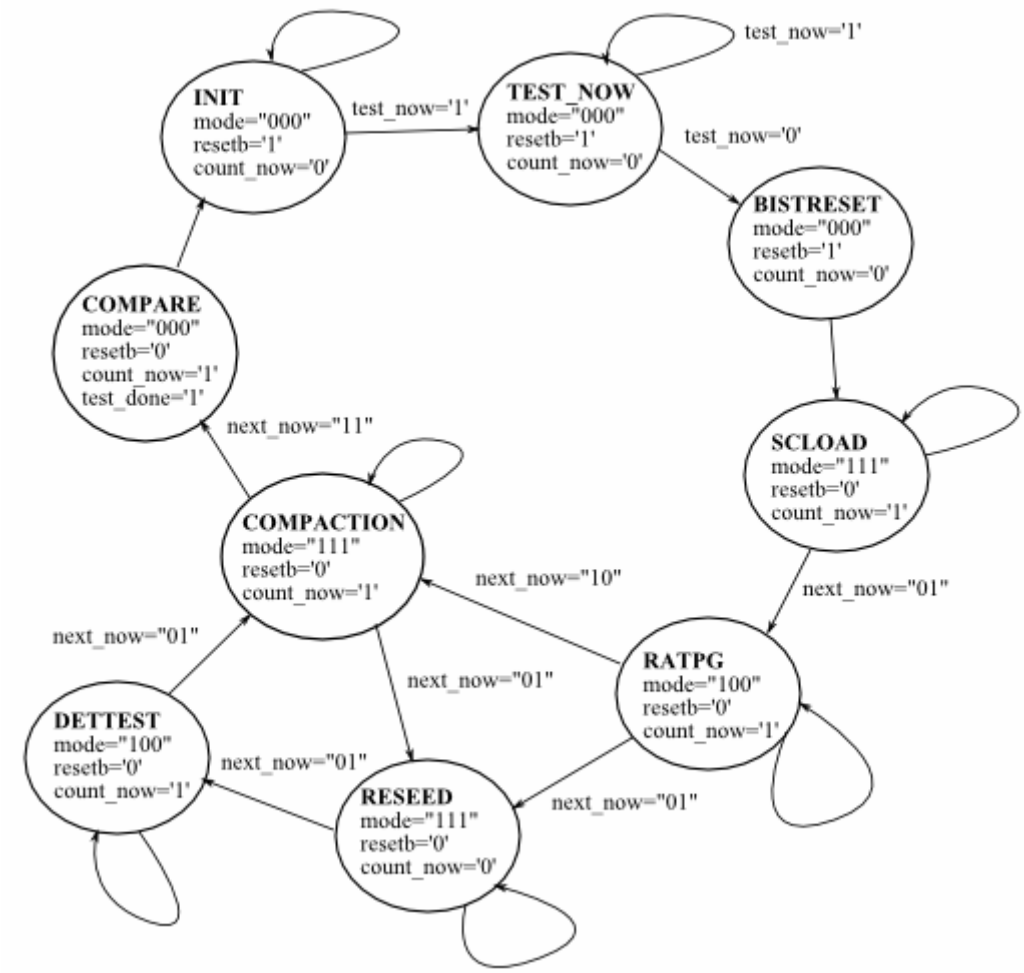
Tato entita obsahuje čítač pro čítaní počtu taktů testu. Čítání je aktivní na signál *count_now* z řadiče. V požadovaných taktech vyšle signál *read* pro načtení nového seedu z paměti.

Control

Radič obvodu, má za úkol obsluhu testování. Graf přechodů řadiče je na obrázku 4.12. Testovaný obvod pracuje v normálním režimu a řadič čeká ve stavu Init dokud nepřijde signál test_now, který přepne do stavu Test_now. V tom čeká na uvolnění tlačítka, potažmo signálu pro testování a poté přejde do stavu Bistreset, ve kterém resetuje BIST a všechny jeho součásti a na další hodinový signál započne testování – přechod do dalšího stavu.

Test začíná stavem Scload, ve kterém je do vstupního a vnitřního scan-chainu nahrán počáteční stav. Délka jeho taktů je rovna počtu PI + PPI. Na určitou hodnotu signálu *next_now* z Counteru, který obstarává toto počítání se posouvá do stavu Ratpg. Zde se provádí počáteční pseudonáhodná fáze testovaní po zadaný počet taktů. Pokud je počet reseedů zvolen nula, přechází se rovnou do stavu Compaction. Pokud ne provede se pro každý reseed cyklus stavů Reseed->Dettest->Compaction. V těch se postupně nejprve nahraje nový seed z paměti (stav Reseed, počet taktů PI+ PPI), dále se provedou požadované takty testu po reseedu (stav Dettest, počet taktu zadán uživatelem) a na konec se odezva vysune do kompaktoru odezvy (stav Compaction, počet taktů PO + PPO).

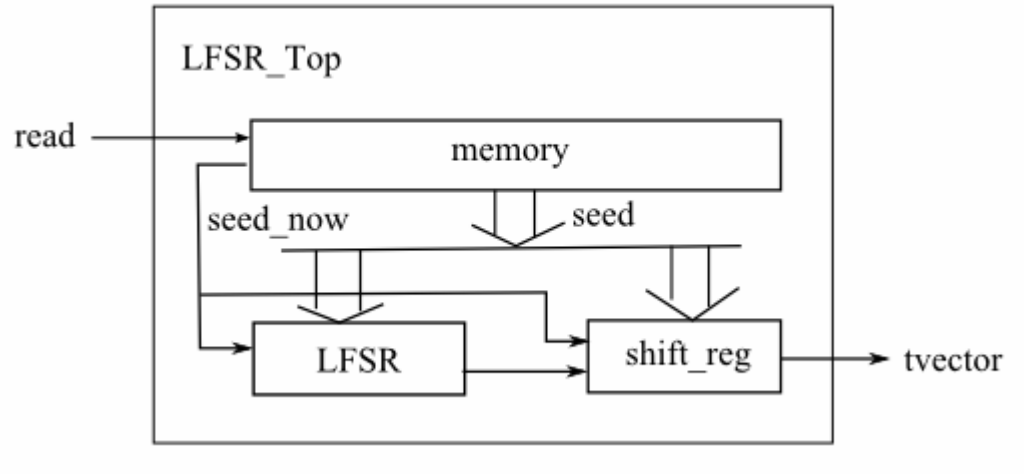
Na konci posledního cyklu reseedu hodnota *next_now* “11“ signalizuje přesun do stavu Compare, ve kterém se porovná hodnota výsledného příznaku s očekávanou hodnotou. Výsledkem jsou hodnoty signálů *bist_done* a *bist_fail*.




Obrázek 4.12 Graf přechodů řadiče pro metodu test-per-scan

LFSR_Top

Entita, jež v sobě obsahuje LFSR, představující TPG, nastavený Shift registrem. A dále paměť. Výstupem je sériově produkovaný testovací vektor. Schéma zapojení entity je na obrázku 4.13.




Obrázek 4.13 Zapojení entity LFSR_Top pro metodu test-per-chain

LFSR a shift_reg

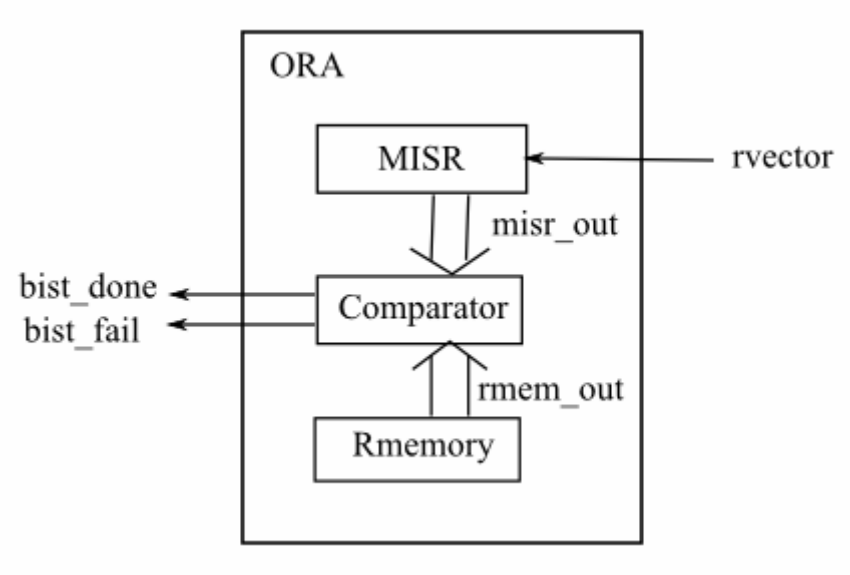
LFSR má délku zadánu parametry ze zdrojového souboru (volitelná). Je-li počet vstupů testovaného obvodu větší, je nastaven shift registrem. To je nutné z důvodu možnosti nahrání celé délky seedu. Jeho zpětná vazba je stejně tak zapojena dle zvolení uživatelem. O zapojení LFSR je více pojednáno v kapitole tomu věnované, viz. předchozí. Na vstupech do jednotlivých buněk LFSR a stejně tak shift registru je umístěn multiplexor, který na aktivní signál *seed_now* přepíná vstup z předchozí buňky (standardní režim generování testovacích vektoru) na vstup z paměti seedu. Ten je vždy nahrán v jednom taktu paralelně z paměti seedu.

Memory

Paměť seedů. Na aktivní signál *read* pošle na sběrnici *seed* následující seed a zaktivní signál *seed_now*, takže seed je načten do LFSR a shift registru.

ORA

Neboli Output Response Analysator. Má za úkol přijímat na vstupu sériově výstup z výstupního scan-chainu a komprimovat jej v 8-mi bitovou odezvu. Dále pak po ukončení testu porovnává hodnotu v příznaku s očekávanou hodnotou v paměti. Schéma zapojení na obrázku 4.6. Skládá se z několika komponent.




Obrázek 4.14 Zapojení komponenty ORA pro metodu test-per-chain

MISR

Přijímá na vstupu sériově výstup z výstupního scan-chainu a komprimuje jej v 8-mi bitovou odezvu. Komprese je obstarávaná blokem MISR. Což je 8-mi bitový LFSR s primitivní zpětnou vazbou.

Comparator

Na aktivní signál *bist_done* porovná hodnotu vystupující z MISRu s hodnotou v Rmemory a vyhodnocení vyšle na signál *bist_fail*, kdy 1 znamená, že test neodhalil žádnou poruchu.

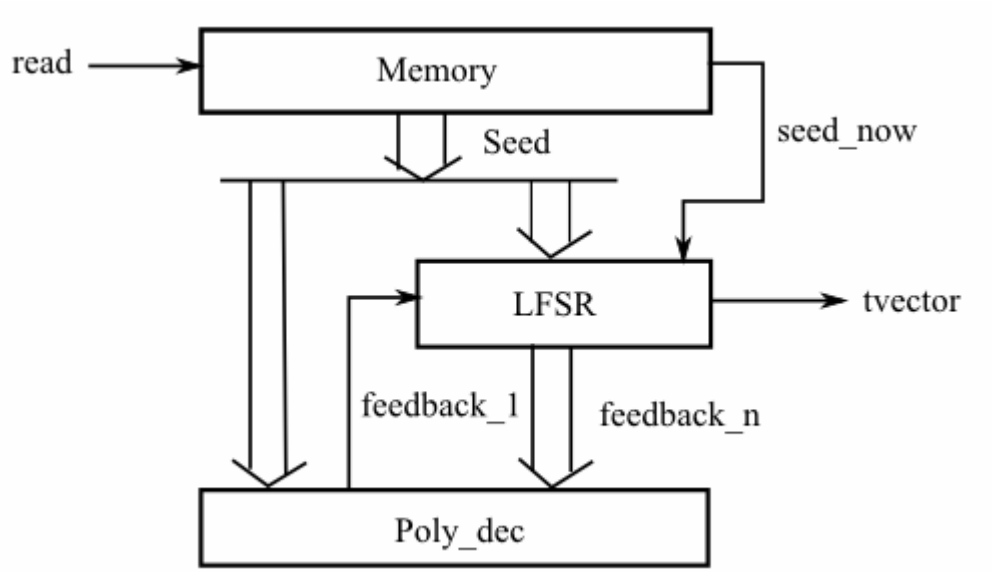
Rmemory

Paměť obsahující hodnotu (8b), kterou očekáváme na konci testu v kompresoru odezvy. Ta je zadávána uživatelem.

### 4.3 Multi-polynomiální LFSR s reseedingem

Struktura BISTu se pro multi-polynomiální LFSR příliš nemění. Pro obě použité metody se změní pouze struktura bloku LFSR_Top.

Seedy uložené v paměti jsou rozšířeny o bity určující zpětnou vazbu. Tyto bity jsou uloženy na nejvyšších bitech seedu a jejich počet je určen počtem různých zpětných vazeb. Ty jsou pak jako vstupy nové komponenty Poly_dec. Propojení je na obrázku 4.15




Obrázek 4.15 Zapojení multi-polynomiálního LFSR

Dále jsou jejími vstupy paralelní výstupy z LFSR a výstupem je jeden bit vstupující do LFSR jako jeho zpětná vazba. Poly_dec dekóduje vstupní bity z paměti při načtení nového seedu a přiřadí LFSR příslušnou zpětnou vazbu, podle žádaného polynomu.

# 5. Testování

V této sekci jsou prezentovány výsledky testování funkčnosti generovaných BISTů, jejího ověřování. Diagnostické schopnosti prezentovaného algoritmu, porovnání. A dále pak výsledky testování vlastností jednotlivých metod implementových BISTů a jejich porovnání z různých hledisek.

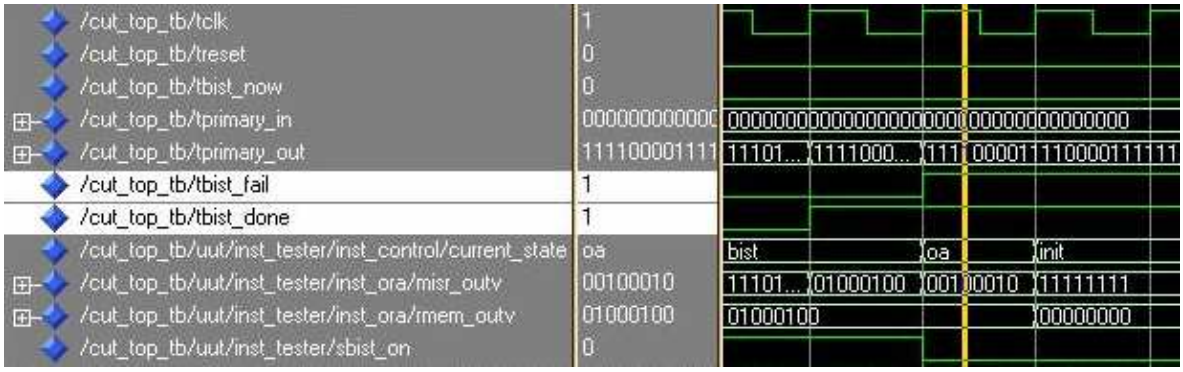
Syntézu do FPGA, kterou jsem zjišťoval velikost jednotlivých bloků nebo celků, jsem prováděl pomocí programu Xilinx ISE 9.1i. Zvolený byl přípravek, který je pro mě k dosažení v hardwarové laboratoři školy FPGA Xilinx Spartan xc2s200e-pq208-6.

K simulaci jsem pak používal program Modelsim XE III 6.0 od firmy Mentor Graphics.

## 5.1 Testování funkčnosti

Testování funkčnosti, tj. ověření, že vytvořené obvody fungují správně, jsem prováděl jak simulací tak i implementací na přípravku v hardwarové laboratoři.

V simulaci jsme nejprve pro vybrané benchmarky (c5315, s713, s9234) provedl simulaci bez poruchy, diky které jsem získal očekávaný příznak na konci testu a ten jsem manuálně zadal do paměti příznaku. Poté jsem provedl znovu bezporuchovou simulaci, abych ověřil správné vyhodnocení testu. Viz obrázek 5.1, kde hodnota signálu bist_fail=’1’ značí shodu s očekávanou hodnotou a tudíž, že test správně neodhalil žádnou poruchu obvodu.

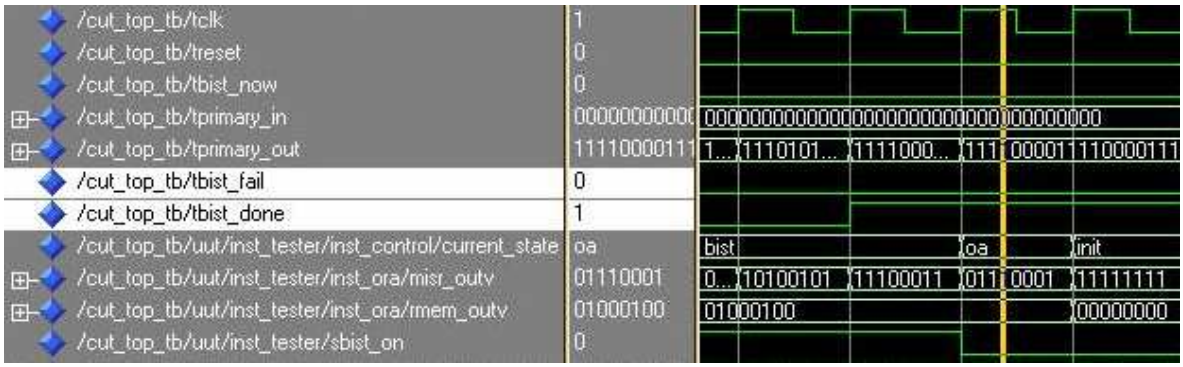



Legenda: hodnota bist_fail=’1’ značí bezporuchový obvod, porovnávána je očekávaná hodnota příznaku – signál rmem_outv s hodnotou v MISRu – signál misr_outv v posledním taktu current_state=bist

Obrázek 5.1 Test obvodu bez poruchy

Poté jsem přímo do VHDL kódu testovaného obvodu implementoval poruchu, jež nemá být daným testem odhalena (informace o odhalených poruchách jsem získal z logfilu programu Hope.exe) a výsledkem byl opět stejný, tj. test poruchu neodhalil.

Naopak pokud jsem zadal poruchu, jež odhalena být má, příznak v MISRu byl na konci testu odlišný od očekávané hodnoty, viz. obrázek 5.2. Test tak tuto poruchu správně odhalil.




Legenda: hodnota bist_fail=’0’ značí poruchový obvod, porovnávána je očekávaná hodnota příznaku – signál rmem_outv s hodnotou v MISRu – signál misr_outv v posledním taktu current_state=bist

Obrázek 5.2 Test obvodu s implementovanou poruchou

Implementaci poruch jsem prováděl přímo úpravou VHDL kódu testovaného obvodu, kdy jsem na signál, jež se měl porouchat přiřadil logickou nulu či jedničku, podle typu poruchy.

Zde uvedené obrázky jsou pro názornost pro relativně malý obvod s713, testování metodou test-per-clock. Toto testování jsem však provedl vždy pro 3 dříve uvedené obvody, pro obě testovací metody.

Pro MPLFSR, obě metody testování, jsem provedl tyto testy obdobně, ovšem z důvodu nutnosti manuálního zadání seedů, jsem je provedl pro menší obvody (c3540, s713).

Pochopitelně správné výsledky jsem nezískal na poprvé, naopak cesta skrze simulaci, byla dlouhá a náročná. Důležité bylo, že jsem si pro každý vytvořený komponent BISTu napsal samostatný testbench, na němž jsem nejprve odladil správnou funkčnost každé komponenty zvlášť. Tyto postupné simulace mi tak umožnili odhalit mnoho chyb již v průběhu práce.

## 5.2 Testování pokrytí poruch

Zde jsou výsledky testování prezentovaného algoritmu pro techniku reseedingu. Testování sem prováděl nad sadou benchmarků ISCAS95. Obvody jak pouze kombinační tak obvody sekvenční. Z daných benchmarků jsem vždy vybral některé a pro ty provedl testování algoritmu pro různě zadané parametry testu.

V této části jsem se zaměřil na testování tohoto algoritmu z hlediska pokrytí poruch, proto zde není podstatná metodu testovaní (o tom až dále) a z toho vyplívající délku testu, nebo velikost zabrané plochy. To je pouze naznačeno atributem pbp – počet bitů paměti, které musíme mít uloženy pro seedy. Výsledky jsou v tabulce 5.1.

Metodou reseedingu by teoreticky bylo možno odhalit pro daný obvod, vždy všechny poruchy, kromě těch nedetekovatelných. To by však vyžadovalo dokonalý nástroj pro generování testovacích vektorů k nepokrytým poruchám. Takový nástroj však nemám k dispozici, Atlanta-M pro některé specifické poruchy obvodu někdy neumí najít příslušný vektor, proto je pro některé benchmarky procento pokrytí nestoprocentní.

Jako základní nastavení jsem vždy používal 700 počátečních taktů LFSR a poté 10 reseedů po 50ti taktech, poté jsem pro složitější obvody přidával nebo ubíral počty reseedů a takty po reseedech, tak abych dosáhl většího pokrytí. Pro malé obvody jsem naopak se snažil pro maximální pokrytí minimalizovat délku testu a počet reseedů. Vlastnosti TPG LFSR byly pro tyto testování vždy shodné.

| obvod | pv | pko | ppt | pr | ptr | ct | pokryti | np | pbp |
|---|---|---|---|---|---|---|---|---|---|
| c432 | 36 | | 700 | 10 | 50 | 1200 | 99.237 | 4 | 360 |
| c499 | 41 | | 700 | 10 | 50 | 1200 | 98.945 | 8 | 410 |
| c880 | 60 | | 700 | 10 | 50 | 1200 | 99.469 | 5 | 600 |
| | 60 | | 700 | 15 | 30 | 1150 | 100 | | 900 |
| c1355 | 41 | | 700 | 10 | 50 | 1200 | 99.492 | 8 | 410 |
| | 41 | | 1100 | 10 | 10 | 1200 | 99.492 | 8 | 410 |
| c1908 | 33 | | 700 | 10 | 50 | 1200 | 98.297 | 39 | 330 |
| | 33 | | 500 | 20 | 30 | 1100 | 98.191 | 34 | 660 |
| | 33 | | 700 | 30 | 20 | 1300 | 99.361 | 12 | 990 |
| | 33 | | 700 | 40 | 15 | 1300 | 99.521 | 9 | 1320 |
| c2670 | 233 | | 700 | 10 | 50 | 1200 | 89.771 | 281 | 2330 |

| obvod | pv | pko | ppt | pr | ptr | ct | pokryti | np | pbp |
|---|---|---|---|---|---|---|---|---|---|
| c2670 | 233 | | 700 | 40 | 10 | 1100 | 94.357 | 155 | 9320 |
| | 233 | | 500 | 70 | 10 | 1200 | 95.668 | 119 | 16310 |
| | 233 | | 500 | 100 | 5 | 1000 | 95.741 | 117 | 23300 |
| | 233 | | 500 | 150 | 10 | 2000 | 95.571 | 117 | 34950 |
| c3540 | 50 | | 500 | 10 | 50 | 1000 | 95.041 | 170 | 500 |
| | 50 | | 700 | 50 | 5 | 1050 | 96.004 | 137 | 2500 |
| | 50 | | 500 | 100 | 5 | 1000 | 96.004 | 137 | 5000 |
| | 50 | | 950 | 50 | 1 | 1000 | 96.004 | 137 | 2500 |
| c5315 | 178 | | 700 | 10 | 50 | 1200 | 98.785 | 65 | 1780 |
| | 178 | | 700 | 20 | 20 | 1100 | 98.897 | 59 | 3560 |
| | 178 | | 1000 | 15 | 10 | 1150 | 98.897 | 59 | 2670 |
| | 178 | | 950 | 50 | 1 | 1000 | 98.897 | 59 | 8900 |
| c6288 | 32 | | 700 | 10 | 50 | 1200 | 99.561 | 34 | 320 |
| | 32 | | 1000 | 15 | 10 | 1150 | 99.561 | 34 | 480 |
| | 32 | | 950 | 50 | 1 | 1000 | 99.561 | 34 | 1600 |
| c7552 | 207 | | 700 | 10 | 50 | 1200 | 96.609 | 256 | 2070 |
| | 207 | | 1000 | 40 | 20 | 1800 | 97.576 | 183 | 8280 |
| | 207 | | 950 | 50 | 1 | 1000 | 97.44 | 193 | 10350 |
| s344 | 9 | 15 | 700 | 10 | 50 | 1200 | 100 | | 240 |
| | 9 | 15 | 100 | 4 | 1 | 104 | 100 | | 96 |
| s713 | 35 | 19 | 700 | 10 | 50 | 1200 | 92.599 | 43 | 540 |
| | 35 | 19 | 500 | 50 | 10 | 1000 | 93.460 | 38 | 2700 |
| | 35 | 19 | 500 | 100 | 5 | 1000 | 93.460 | 38 | 5400 |
| s5378 | 35 | 179 | 700 | 10 | 50 | 1200 | 94.113 | 271 | 2140 |
| | 35 | 179 | 700 | 40 | 10 | 1100 | 95.981 | 185 | 8560 |
| | 35 | 179 | 1000 | 50 | 5 | 1500 | 97.024 | 137 | 10700 |
| | 35 | 179 | 1000 | 100 | 5 | 1500 | 98.132 | 86 | 21400 |
| | 35 | 179 | 1000 | 150 | 1 | 1150 | 99.131 | 40 | 32100 |
| | 35 | 179 | 1000 | 200 | 1 | 1200 | 99.131 | 40 | 42800 |
| s9234 | 19 | 228 | 700 | 10 | 50 | 1200 | 76.397 | 1635 | 2470 |
| | 19 | 228 | 1500 | 20 | 50 | 2500 | 82.373 | 1221 | 4940 |
| s13207 | 31 | 669 | 1500 | 20 | 50 | 2500 | 87.061 | 1270 | 14000 |

Legenda: pv –počet vstupů, pko – počet klopných obvodů, ppt – počet počátečních taktů, pr- počet reseedů, ptr – počet taktů reseedu, ct – celkem taktů, np – nepokryté poruchy,
pbp- počet bitů paměti

Tabulka 5.1 Pokrytí testem technikou reseedingu

Další Tabulka 5.2 ukazuje porovnání techniky reseedingu s obyčejným LFSR.

Pro zvolené benchmarky, při stejném počtu testovacích vektorů, je v tabulce porovnáno pokrytí poruch daným testem.

Jak jsme očekával procento pokrytí poruch je pro techniku reseedingu výrazně vyšší, než pro obyčejné LFSR a dovoluje nám dosáhnout takřka sto procentního pokrytí (omezení jsem zmínil dříve).

| | | | LFSR s reseedingem | | | | | LFSR | |
|---|---|---|---|---|---|---|---|---|---|
| **obvod** | **pv** | **pko** | **ppt** | **pr** | **ptr** | **ct** | **pokryti** | **ct** | **pokryti** |
| c432 | 36 | | 700 | 10 | 50 | 1200 | 99.237 | 1200 | 99.237 |
| c880 | 60 | | 700 | 15 | 30 | 1150 | 100 | 1150 | 98.726 |
| c1908 | 33 | | 700 | 40 | 15 | 1300 | 99.521 | 1300 | 96.860 |
| c3540 | 50 | | 950 | 50 | 1 | 1000 | 96.004 | 1000 | 94.545 |
| c5315 | 178 | | 700 | 20 | 20 | 1100 | 98.897 | 1100 | 98.523 |
| c6288 | 32 | | 700 | 10 | 50 | 1200 | 99.561 | 1200 | 99.561 |
| c7552 | 207 | | 700 | 10 | 50 | 1200 | 96.609 | 1200 | 94.821 |
| | 207 | | 1000 | 40 | 20 | 1800 | 97.576 | 1800 | 95.099 |
| s713 | 35 | 19 | 700 | 10 | 50 | 1200 | 92.599 | 1200 | 91.222 |
| | 35 | 19 | 500 | 50 | 10 | 1000 | 93.460 | 1000 | 91.050 |
| s5378 | 35 | 179 | 1000 | 50 | 5 | 1500 | 97.024 | 1500 | 95.568 |
| | 35 | 179 | 1000 | 200 | 1 | 1200 | 99.131 | 1200 | 94.873 |
| s9234 | 19 | 228 | 700 | 10 | 50 | 1200 | 76.397 | 1200 | 75.011 |
| | 19 | 228 | 1500 | 20 | 50 | 2500 | 82.373 | 2500 | 79.125 |
| s13207 | 31 | 669 | 1500 | 20 | 50 | 2500 | 87.061 | 2500 | 83.566 |

Legenda: pv –počet vstupů, pko – počet klopných obvodů, ppt – počet počátečních taktů, pr- počet reseedů, ptr – počet taktů reseedu, ct – celkem taktů

Tabulka 5.2 Porovnání techniky reseedingu s obyčejným LFSR

## 5.3 Testování vlastností BIST

Hlavními testovanými vlastnostmi, které určují kvalitu BISTu jsou plocha na čipu, délka testu a pokrytí poruch daným testem. Na tyto 3 základní charakteristiky jsem se proto zaměřil při testování a porovnávání jednotlivých metod a technik BISTu, jež jsme implementoval.

### 5.3.1 Komponenty BIST

Nejprve jsem se zaměřil na testování velikostí bloků BISTu a porovnával je pro obě použité testovací metody.

Blok s řadičem - Control, se pro různé obvody či parametry nemění. Porovnání pro různé metody je v tabulce 5.3. Metoda test-per-scan má náročnější řadič, z toho vyplívá větší zabraná plocha.

| | test-per-clock | | | test-per-scan | | |
|---|---|---|---|---|---|---|
| komponenta | slices | slice FF | 4LUTs | slices | slice FF | 4LUTs |
| řadič (Control) | 3 | 3 | 6 | 8 | 4 | 14 |

Tabulka 5.3 Velikosti komponenty řadiče

Blok řídící reseeding, čítač se mění v závisti na vlastnostech testovaného obvody, počtu taktů a počtu reseedů a jejich délce. V BISTu metodou test-per-clock je zastoupen blokem Controller, v metodě test-per-scan pak blokem Counter. Porovnání

pro různé metody  a je v tabulce 5.4. Toto měření jsme prováděl pro jeden obvod, avšak s třemi různými nastaveními parametrů reseedingu. Jak naznačuje tabulka pro metodu test-per scan je čítač náročnější na plochu, neboť operace se scan-chainem vyžadují více taktů a přepínání několika módů funkce.

| | test-per-clock | | | test-per-scan | | |
|---|---|---|---|---|---|---|
| komponent/ taktů a reseedů | slices | slice FF | 4LUTs | slices | slice FF | 4LUTs |
| čítač (Controller, Counter) | | | | | | |
| 1000 | 57 | 33 | 104 | 61 | 35 | 113 |
| 700 + 30*10 | 67 | 34 | 122 | 82 | 35 | 153 |
| 950 + 50*1 | 61 | 34 | 111 | 94 | 35 | 127 |

Tabulka 5.4 Velikosti komponent čítače

Blok pro kompresi odezvy je u metody test-per-scan realizován vždy 8-mi bitovým MISRem se sériovým vstupem, naproti tomu u druhé metody je tento MISR nutno nastavit shift-registrem pro paralelní vstup. Tudíž pro obvody s velkým počtem výstupů (PO + PPO) je u této metody blok ORA vždy výrazně větší. Srovnání velikostí celého bloku ORA, včetně paměti příznaku a komparátoru, pro různý počet výstupů CUT je v tabulce 5.5.

| | test-per-clock | | | test-per-scan | | |
|---|---|---|---|---|---|---|
| Komponenta/ CUT výstupů | slices | slice FF | 4LUTs | slices | slice FF | 4LUTs |
| Ora | | | | | | |
| 8 CUT outputs | 3 | 2 | 6 | 3 | 2 | 6 |
| 32 CUT outputs | 21 | 2 | 36 | 3 | 2 | 6 |
| 128 CUT outputs | 76 | 2 | 132 | 3 | 2 | 6 |

Tabulka 5.4 Velikosti komponent ORA

Blok LFSR nastavený shift-registrem a velikost paměti seedů jsou pro obě metody vždy shodné, v závislosti na vstupních parametrech. Tudíž i plocha zabraná těmito komponentami je pro obě metody shodná.

Overhead pro metodu  test-per-clock navíc naroste o multiplexor na primárních vstupech testovaného obvodu.

Naopak zabraná plocha pro metodu test-per-scan naroste o vstupní a výstupní scan-chainové buňky.

Pro obyčejný LFSR bez reseedingu ubude výrazně overhead o paměť reseedu, porovnání v následující sekci

U Multi-polynomiálního LFSR s reseedingem však bude blok LFSR_Top větší o blok Poly_dec pro přepínání zpětné vazby a zvětší se obsah paměti seedů. Naopak ubude shift-register pro metodu test-per-scan. Porovnání v tabulce 5.5. Mimo tyto blok pak zůstává zabraná plocha pro multi-polynomiální LFSR stejná.

| | metoda test-per-scan, 32b LFSR, 5 reseedů | | | | | |
|---|---|---|---|---|---|---|
| | LFSR s reseedingem | | | MPLFSR s reseedingem | | |
| komponenta/pro obvod | slices | slice FF | 4LUTs | slices | slice FF | 4LUTs |
| LFSR_Top | | | | | | |
| c5315 | 97 | 188 | 179 | 69 | 70 | 126 |
| s713 | 81 | 98 | 149 | 69 | 70 | 126 |
| s5378 | 161 | 258 | 307 | 69 | 70 | 126 |

Tabulka 5.5 Velikosti komponenty LFSR_Top

### 5.3.2 Celý obvod

V této sekci prezentuji výsledky syntézy pro různé obvody s BISTem a porovnání pro obě testovací metody. Testované obvody jsou vybrané benchmarky ISCAS95. Parametry pro test jsou vždy uvedené v tabulce před výsledkami zabrané plochy FPGA. Nastavení LFSR jsme zvolil konstantně na 32 bitů s primitivní zpětnou vazbou. Výsledky jsou v tabulce 5.6.

| | test-per-clock | | | | | test-per-scan | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| obvod | slices | slice FF | 4LUTs | taktů | delay | slices | slice FF | 4LUTs | taktů | delay |
| c2670 | pv: 233, ct: 700 + 40*10, pokrytí: 94.357 | | | | | | | | | |
| | 751 | 532 | 1177 | 1100 | 21.449 | 806 | 770 | 1199 | 16292 | 21.752 |
| c3540 | pv: 50, ct: 950 + 50*1, pokrytí: 96,004 | | | | | | | | | |
| | 505 | 185 | 862 | 1000 | 31.570 | 614 | 242 | 1060 | 4749 | 32.426 |
| c5315 | pv: 178, ct: 700 + 10*50, pokrytí: 98.785 | | | | | | | | | |
| | 707 | 411 | 1257 | 1200 | 24.416 | 797 | 594 | 1366 | 4397 | 24.238 |
| c5315 | pv: 178, ct: 1000 + 15*10, pokrytí: 98.897 | | | | | | | | | |
| | 703 | 408 | 1252 | 1150 | 24.416 | 820 | 590 | 1413 | 5857 | 25.303 |
| c7552 | pv: 207, ct: 1000 +40*20, pokrytí: 97.576 | | | | | | | | | |
| | 1037 | 506 | 1672 | 1800 | 29.558 | 1087 | 713 | 1755 | 14646 | 29.354 |
| s713 | pv: 35, pko: 19, ct: 700 + 10*50, pokrytí: 92.599 | | | | | | | | | |
| | 259 | 206 | 462 | 1200 | 22.065 | 288 | 231 | 505 | 2223 | 23.441 |
| s5378 | pv: 35, pko: 179, ct: 700 + 10*50, pokrytí: 94.113 | | | | | | | | | |
| | 781 | 711 | 1364 | 1200 | 20.440 | 637 | 588 | 1172 | 5843 | 19.107 |
| s5378 | pv: 35, pko: 179, ct: 1000 + 50*5, pokrytí: 97.024 | | | | | | | | | |
| | 890 | 756 | 1544 | 1250 | 20.759 | 741 | 633 | 1344 | 23613 | 19.107 |
| s9234 | pv: 19, pko: 228, ct: 700 + 10*50, pokrytí: 76.397 | | | | | | | | | |
| | 934 | 808 | 1611 | 1200 | 10.655 | 737 | 633 | 1341 | 6426 | 11.125 |
| s13207 | pv: 31, pko: 669, ct: 1500 + 20*50, pokrytí: 87.061 | | | | | | | | | |
| | 1920 | 2180 | 3362 | 2500 | 18.853 | 1437 | 1704 | 2547 | 33019 | 17.265 |

Legenda: pv –počet vstupů, pko – počet klopných obvodů, ct – celkem taktů

Tabulka 5.6 Velikosti obvodů s BIST

Jak je pozorovatelné z grafu tak výsledky se pro obě metody značně liší. Pro kombinační obvody (benchmarky s prefixem c) se jeví jako výrazně výhodnější metoda testovaní test-per-clock. Vykazuje nižší nároky na zabranou plochu a také počet taktů je menší.

U obvodů sekvenčních je pro metodou test-per-scan stále několikanásobně vyšší počet taktů, nicméně pro zvětšující se velikost a náročnost obvodu se oproti metodě test-per-clock výrazně snižuje zabraná plocha.

Tyto výsledky jsme vzhledem k prezentované struktuře implementace obou metod očekával. Testování metodou test-per-clock je výrazně rychlejší. Vyplatí se především pro menší obvody. Pro obvody velké má však velké nároky na propojení vstupů a výstupů. Tam se mi jeví výhodnější metoda test-per-scan, kde za cenu delšího testu lze získat výraznou úsporu na zabrané ploše.

Další Tabulka 5.7 ukazuje porovnání zabrané plochy, pro vybrané obvody, pro LFSR s reseedingem oproti obyčejnému LFSR.

| | metoda test-per-scan | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | LFSR s reseedingem | | | | | LFSR | | | | |
| obvod | slices | slice FF | 4LUTs | delay | pokrytí | slices | slice FF | 4LUTs | delay | pokrytí |
| c5315 | pv: 178, ct: 1000 + 15*10 | | | | | pv: 178, ct: 1150 | | | | |
| | 820 | 590 | 1413 | 25.303 | 98.897 | 688 | 529 | 1017 | 24.900 | 98.598 |
| s713 | pv: 35, pko: 19, ct: 700 + 10*50 | | | | | pv: 35, pko: 19, ct: 1200 | | | | |
| | 288 | 231 | 505 | 23.441 | 92.599 | 191 | 179 | 284 | 21.351 | 91.222 |
| s5378 | pv: 35, pko: 179, ct: 1000 + 50*5 | | | | | | | | | |
| | 741 | 633 | 1344 | 19.107 | 97.024 | 521 | 531 | 778 | 19.107 | 95.068 |
| s9234 | pv: 19, pko: 228, ct: 700 + 10*50 | | | | | | | | | |
| | 737 | 633 | 1341 | 11.125 | 76.397 | 627 | 568 | 925 | 11.125 | 75.011 |

Legenda: pv –počet vstupů, pko – počet klopných obvodů, ct – celkem taktů

Tabulka 5.7 Velikosti obvodů s BIST 2

Je vidět, že zvýšení pokrytí pomocí reseedingu je za cenu zvýšení většího zpoždění obvodu a větších nároků na zabranou plochu.

# 6. Závěr

Řešení této práce pro mě nebylo vstupem na zcela neznámou půdu, neboť dané problematice jsem se věnoval již v předchozích samostatných či týmových pracích. Nicméně i tak si vyžádalo značné množství konzultací a nastudování materiálů a prací od renomovaných expertů na tuto problematiku.

Myslím si, že toto řešení, k němuž jsem došel, splňuje vytyčené cíle práce a představy, nebo se jim přinejmenším velmi přibližuje. Podařilo se mi vytvořit programy pro generování VHDL kódu BISTu integrovaného do obvodu zadaného netlistem. Implementovat algoritmus pro techniku reseedingu a to pro obě požadované metody testu.

V sekci testování pak nabízím srovnání testovaných parametrů na zkušebních obvodech pro tyto různé metody, včetně jejich porovnání s obyčejným LFSR či rozšířením na multi-polynomiální LFSR s reseedingem. Funkci těchto obvodů jsme dále ověřil jak za pomocí simulací, tak pomocí implementace do FPGA.

Velmi kladně také hodnotím přínos práce pro mne samotného. Umožnila mi hlouběji
prozkoumat oblast testování pomocí BISTů a také v procvičit v implementačních prostředcích jako je například jazyk VHDL.

# 7. Seznam použitých zkratek

ATPG (***A**utomatic **T**est **P**attern **G**enerator*) – automatický generátor testovacích vzorků
BIST (***B**uilt-**I**n **S**elf-**T**est*) – vestavěné diagnostické prostředky
CUT (***C**ircuit **U**nder **T**est*) – testovaný obvod
FPGA (***F**ield **P**rogrammable **G**ate **A**rrays*) – programovatelná hradlová pole
LFSR (***L**inear **F**eedback **S**hift **R**egister*) – lineární zpětnovazební posuvný registr
LUT (***L**ook-**U**p **T**able*) – tabulka pomocí níž se v FPGA generují logické funkce
MISR (***M**ultiple-**I**nput **S**ignature **R**egister*) – vícevstupový příznakový registr
PI (***P**rimary **I**nputs*) –primární vstupy testovaného obvodu
PPI (***P**seudo-**P**rimary **I**nputs*) – pseudo-primární vstupy testovaného obvodu
PO (***P**rimary **O**utputs*) –primární výstupy testovaného obvodu
PPO (***P**seudo-**P**rimary **O**utputs*) – pseudo-primární výstupy testovaného obvodu
PRPG (***P**seudo-**R**andom **P**attern **G**enerator*) – pseudonáhodný generátor testovacích vzorků
OA (***O**utput **A**nalysis*) – příznaková analýza
ORA (***O**utput **R**esponse **A**nalyser*) – příznakový analyzátor odezev
TPG (***T**est **P**attern **G**enerator*) – generátor testovacích vzorků
VHDL (***V**ery **H**igh **D**escription **L**anguague*) – jazyk pro popis struktury a chování obvodů na vyšší úrovni
VLSI (***V**ery **L**arge **S**cale **I**ntegration*) – velmi vysoký stupeň integrace

# 8. Seznam použitých zdrojů


- Prof. Charles E. Stroud – A Designer's Guide to Built-In Self-Test, Stringer 2002

- Prof. Ing. Jan Hlavička, DrSc. – Diagnostika a spolehlivost, skriptum ČVUT vydání druhé, vydalo Vydavatelství ČVUT 1998

- Sybille Hellebrand**,** Janusz Rajski**,** Steffen Tarnick**,** Srikanth Venkataraman and Bernard Courtois - Built-In Test for Circuits with Scan Basedon Reseeding of Multiple-Polynomial Linear Feedback Shift Registers, February 1995

- Sybille Hellebrand, Birgit Reeb, Steffen Tarnick, Hans-Joachim Wunderlich - Pattern Generation for a Deterministic BIST Schneme, University of Siegen, November 1995Germany

- O.Novák, Z.Plíva, J. Nosek, A.Hlawiczka, T Garbolino and K.Gucwa - Test-Per-Clock Logic BIST with Semi-Deterministic Test Patterns and Zero-Aliasing Compactor, 2004

- A. Jutman, A. Tsertov, R. Ubar - Calculation of LFSR Seed and Polynomial Pair for BIST Applications, Tallinn University of Technology

# 9. Přílohy

## 9.1 Obsah přiloženého CD

**9.1 Obsah přiloženého CD**

```
├ index.html - výchozí stránka projektu s odkazy na jeho součásti
├ install.txt - pokyny k instalaci
├ readme.txt - obsah CD
├ [html] - soubory pro html stránky bakalářské práce
├ [programs]
│   ├ [Bistgenr] - soubory k programu bistgenr.exe
│   ├ [Bistgenr_mp] - soubory k programu bistgenr_mp.exe
│   ├ [Cutgen] - soubory k programu cutgen.exe
│   ├ Bistgenr.zip – zabalené soubory k programu bistgenr.exe
│   ├ Bistgenr_mp.zip - zabalené soubory k programu bistgenr_mp.exe
│   └ Cutgen.zip - zabalené soubory k programu cutgen.exe
├ [text] - obsahuje vlastní text BP
│   ├ BP-RobekJiri.doc - text BP ve formátu Microsoft Office
│   └ BP-RobekJiri.pdf - text BP v univerzálním formátu pdf
└ obsah_cd.zip - kompletní obsah cd v jednom balíčku
```

## 9.2 Manuál programu Bistgenr.exe

**PROGRAM bistgenr.exe – manual**

Program bistgenr.exe generuje, pro obvod zadaný ve formátu .bench, syntetizovatelný VHDL kód reprezentující funkční obvod se začleněným BISTem.

**1. Programy a soubory nutné mit v adresáři s tímto programem pro jeho správnou funkčnost (program bistgen je používá/volá střídavě za běhu)**

atalanta-M.exe - generování testovacích vektorů pro nepokryté poruchy
cygwin1.dll - nutný pro chod programu hope a atalanta
hope.exe - testování pokrytí poruch a generování seznamu nepokrytých poruch
lfsr.exe - simuluje beh lfsr po reseedu, vybírá nejlepší seed
lfsrbegin.exe - simuluje počáteční běh lfsr do prvního reseedu
parameters.txt - soubor se vstupnimy parametry
vhdlgen_r_si.exe - generování VHDL testeru, pro metodu test-per-clock
vhdlgen_r_sc.exe - generování VHDL testeru, pro metodu test-per-scan
<obvod>.bench - bench soubor kombinačního obvodu, pro který chceme generovat VHDL

**2. Spouštění programu**

předpis: bistgenr.exe obvod metoda (prepínač)

argumenty - povinné:

obvod název obvodu, pro který chceme generovat VHDL - pouze název obvodu, nikoliv název zdrojového souboru, který se však musí jmenovat stejně (max string délka 20)
metoda -tpc pro volbu metody test-per-clock
-tps pro volbu metody test-per-scan

argument - nepovinný:

přepínač -m pokud chceme načíst seedy ze souboru parametres.txt

příklad: bistgenr.exe c432 -tps -m
- vygeneruje VHDL CUT+BIST pro obvod c432, metodou test-per-scan, parametry jsou načteny ze souboru parameters.txt

**2. Vstupy a výstupy**

vstupy: <obvod>.bench
- soubor s popisem obvodu pomocí netlistu
- pro správný běh programu je nutná koncovka .bench

parameters.txt
- soubor obsahující vstupni parametry testeru

- každý údaj se požaduje být uvozen dvojtečkou

delka LFSR(pocet bitu) - počet bitu generujícího LFSR
polynom pro zpetnou vazbu LFSR - polynom pro zpetnou vazbu LFSR, zadán binárně, kde jednička značí zapojení do zpětné vazby, nejvyšší bit x na n je nejvíce vpravo, člen x na 0 čili +1 není zahrnut a je započítán automaticky
př :100011 == x6+x5+x1+1
počet počátečních taktů LFSR - počet taktů, po který TPG poběží, než dojde k prvnímu reseedu (max 5000)
počet reseedů - počet reseedů obvodu (max 100)
počet taktů po každém reseedu - počet taktů, po který TPG poběží, po každém reseedu(max 5000)
přiznak pro srovnání a vyhodnocení testu (8b) -příznak očekávaný na konci testu pro srovnání s obsahem kompresoru odezev po ukončení testu
reseedy(nacteny, pokud je aktivni prepinac -m) - seedy pro manuální zadání, binárně

pro správné načtení parametrů se požaduje aby byl uvozen dvojtečkou
příklad obsahu souboru parametres.txt:

```
delka LFSR(pocet bitu)
:6

polynom pro zpetnou vazbu LFSR
:100001

pocet pocatecnich taktu LFSR
:20

pocet reseedu
:5

pocet taktu po kazdem reseedu
(0 - bude aplikovan pouze seed)
:0

priznak pro srovnani a vyhodnoceni testu (8 bitu)
:10101001

reseedy(nacteny, pokud je aktivni prepinac -m)
:000000000000000111111111
:000000000000000011111111
:000000000001111111111111
:000011111110000000001111
:111100000000000000001111
```

výstupy: VHDL soubory <obvod><část obvodu>.vhd

(např. c432Tester.vhd)
soubor <obvod>CUT_Top.vhd je top vhdl soubor celé struktury
-> jeho vstupy : primary_in - primarní vstupy CUT
clk, reset - hodiny, reset
bist_now - signál pro spuštění testu
-> jeho výstupy : primary_out - primarní výstupy CUT
bist_fail - logická '1' značí shodu příznaku s očekávanou hodnotou
bist_done - logická '1' dokončený test

informační výstupní soubory:
<obvod>.lgc - hope logfile, informace o dosaženém okrytí poruch
vektors.vec - použité testovací vektory
reseeds.vec - použité reseed vektory
<obvod>.uf1 - nepokryté poruchy

## 9.3 Manuál programu Bistgenr_mp.exe

**PROGRAM bistgenr_mp.exe – manual**

Program bistgenr.exe generuje, pro obvod zadaný ve formátu .bench, syntetizovatelný VHDL kód reprezentující funkční obvod se začleněným BISTem.

1. Spouštění programu

předpis: bistgenr_mp.exe obvod metoda

argumenty - povinné:

obvod  název obvodu, pro který chceme generovat VHDL - pouze název obvodu, nikoliv název zdrojového souboru, který se však musí jmenovat stejně (max string délka 20)
metoda  -tpc  pro volbu metody test-per-clock
-tps  pro volbu metody test-per-scan

příklad: bistgenr_mp.exe c432 -tps
- vygeneruje VHDL CUT+BIST pro obvod c432, metodou test-per-scan

3. Vstupy a výstupy

vstupy:  <obvod>.bench
- soubor s popisem obvodu pomocí netlistu
- pro správný běh programu je nutná koncovka .bench

parameters.txt
- soubor obsahující vstupni parametry testeru
- každý údaj se požaduje být uvozen dvojtečkou

delka LFSR(pocet bitu) - počet bitu generujícího LFSR
počet počátečních taktů LFSR - počet taktů, po který TPG poběží, než dojde k prvnímu reseedu (max 5000)
polynom pro zpetnou vazbu LFSR - polynom pro zpetnou vazbu LFSR, zadán binárně, kde jednička značí zapojení do zpětné vazby, nejvyšší bit x na n je nejvíce vpravo, člen x na 0 čili +1 není zahrnut a je započítán automaticky
př :100011  ==  x6+x5+x1+1
přiznak pro srovnání a vyhodnocení testu (8b) -příznak očekávaný na konci testu pro srovnání s obsahem kompresoru odezev po ukončení testu
počet reseedů - počet reseedů obvodu (max 100)
reseedy - manuálně zadané seedy a příslušné polynomy
první číslo je binárně zadaná zpětná vazba, formát viz předchozí
druhé číslo je samotný seed, který bude načten
třetí číslo je počet taktů po tomto reseedu

pro správné načtení parametrů se požaduje aby byl uvozen dvojtečkou
příklad obsahu souboru parametres.txt:

```
delka LFSR
:6

pocet pocatecnich taktu LFSR
:100

polynom pro zpetnou vazbu LFSR
:100001

priznak pro srovnani a vyhodnoceni testu (8 bitů)
:10101001

pocet reseedu
:4

reseedy (polynom + seed)
:100001 001111 0
:110101 100110 2
:101101 110011 0
:100001 011001 5
```

výstupy: VHDL soubory <obvod><část obvodu>.vhd
(např. c432Tester.vhd)
soubor <obvod>CUT_Top.vhd je top vhdl soubor celé struktury
-> jeho vstupy : primary_in - primarní vstupy CUT
clk, reset - hodiny, reset
bist_now - signál pro spuštění testu
-> jeho výstupy : primary_out - primarní výstupy CUT
bist_fail - logická '1' značí shodu příznaku s očekávanou hodnotou
bist_done - logická '1' dokončený test