ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Návrh TSC obvodů v FPGA

2004 Leoš Kafka

ANOTACE

Tato diplomová práce se zabývá návrhem úplně samočinně kontrolovaných obvodů (TSC) v hradlových polích FPGA. V první části práce jsou podrobně popsány dva způsoby návrhu TSC obvodů. V druhé části jsou popsány nástroje pro ověření TSC vlastnosti obvodu, které byly vyvinuty v rámci této práce. V poslední části jsou popsány výsledky experimentů pro několik vzorových obvodů.

ABSTRACT

This thesis deals with the design of the totally self-checking circuits (TSC) in the Field Programmable Gate Arrays (FPGA). Two ways of the design of TSC are described in details in the first part of my work. The tools for TSC property checking are described in the second part. These tools were developed as a part of my work. The last part presents results of some experiments for several benchmarks.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) 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 17. května 2004

….....

Leoš Kafka

Obsah

1 Úvod

V této práci se budeme zabývat návrhem samočinně kontrolovaných obvodů (TSC). Tyto obvody mají tu vlastnost, že jejich výstupy jsou neustále kontrolovány a případná chyba je okamžitě detekována. Druhou vlastností je schopnost samočinného testování. U TSC obvodů je tak zajištěna integrita dat a tyto obvody navíc nepotřebují periodickou diagnostiku.

Budeme se zabývat návrhem TSC obvodů v hradlových polích FPGA. Hradlové pole lze snadno nakonfigurovat nahráním dat do obvodu a vývojový cyklus návrhu tak může být mnohem kratší a méně nákladný než v případě zákaznických obvodů (ASIC). Změny v návrhu obvodu způsobené opravou chyby v návrhu nebo změnou požadavků zákazníka také nepřináší další náklady. Nevýhodou je vyšší cena obvodu. Pro vývoj prototypů a výrobu malosériových zařízení jsou ale cenově výhodnější než zákaznické obvody. Podstatnou výhodou FPGA je možnost (i částečné) změny konfigurace obvodu za běhu systému.

Zaměříme se hlavně na sekvenční obvody. Nejprve popíšeme a podrobně rozebereme již publikované způsoby návrhu TSC obvodů v FPGA. Dalším krokem bude návrh programových nástrojů, které nám umožní vyzkoušet návrh TSC obvodů v praxi. Zaměříme se na návrh sekvenčních obvodů využívající bezpečnostní kódy. Budeme proto potřebovat nástroj pro zakódování vstupů, výstupů a vnitřního stavu obvodu, nástroj pro převod do formátu VHDL, který můžeme zpracovat nástrojem XILINX ISE pro syntézu obvodu do FPGA a simulátor poruch, který nám umožní vkládat poruchy na úrovni popisu EDIF a sledovat jejich projevy. Posledním krokem bude ověření vlastností takto zabezpečeného obvodu na vybraných benchmarcích.

Rozdělení tohoto dokumentu je následující:

V kapitole 2 je uveden teoretický úvod do této problematiky. Je zde popsán princip funkce hradlových polí, dále popis, rozdělení a vlastnosti různých bezpečnostních kódů, modely poruch v logických obvodech a speciálně v hradlových polích, způsoby návrhu obvodů bezpečných proti poruchám a nakonec princip úplně samočinně kontrolovaných obvodů.

V kapitole 3 jsou popsány dva způsoby návrhu TSC obvodů v hradlových polích. Jako první je uveden návrh označovaný jako MD-architektura. Obvod podle MD-architektury nepoužívá na výstupu bezpečnostní kódování a kontrolu kódových slov, ale kontroluje, zda je na výstupu správné slovo. Vnitřně pracuje v kódu 1 z N, a to by společně se způsobem implementace mělo zajišťovat TSC vlastnost obvodu. V kapitole 3.1.3 toto tvrzení ověříme a v kapitole 3.1.4 porovnáme velikost takto zabezpečeného obvodu s velikostí původního obvodu.
Druhý způsob návrhu je uveden v kapitole 3.2. Původní popis obvodu je změněn na popis obsahující pouze monotónní funkce. Použitím kódu schopného detekovat jednosměrné chyby je zajištěna

bezpečnost obvodu proti poruchám. Jeden takový kód, označovaný jako redukovaný kód $M_z N$, je popsán v kapitole 3.2.2.

V kapitole 4 jsou popsány vlastní metody a algoritmy použité pro ověřování TSC vlastností obvodů. Kapitola 4.1 obsahuje formáty použitých souborů a způsob překladu do vnitřní formy. Dále je v kapitole 4 uveden algoritmus pro kódování redukovaným kódem $M_z N$, převod z formátu KISS2 do formátu PLA a z formátu PLA do VHDL a způsob generování testu pro sekvenční obvody. V kapitole 4.6 je popsán vlastní simulátor poruch v obvodu, nazvaný FE-SIM. V poslední podkapitole 4.7 je popsáno rozdělení poruch z hlediska on-line kontroly.

V kapitole 5 jsou popsány experimenty společně s jejich výsledky. Zaměříme se na návrh používající bezpečnostní kódy. Nejprve je provedeno porovnání redukovaného kódu $M_z N$ s Bergerovým kódem. Dalším experimentem ověříme, jestli lze použitím bezpečnostního kódu bez přizpůsobení implementace použitému kódu dosáhnout TSC vlastnosti obvodu. V posledním experimentu ověříme, jestli způsob návrhu popsaný v kapitole 3.2 umožňuje dosáhnout TSC vlastnosti obvodu.

V kapitole 6 je zhodnocení dosažených výsledků a návrhy na budoucí práci.

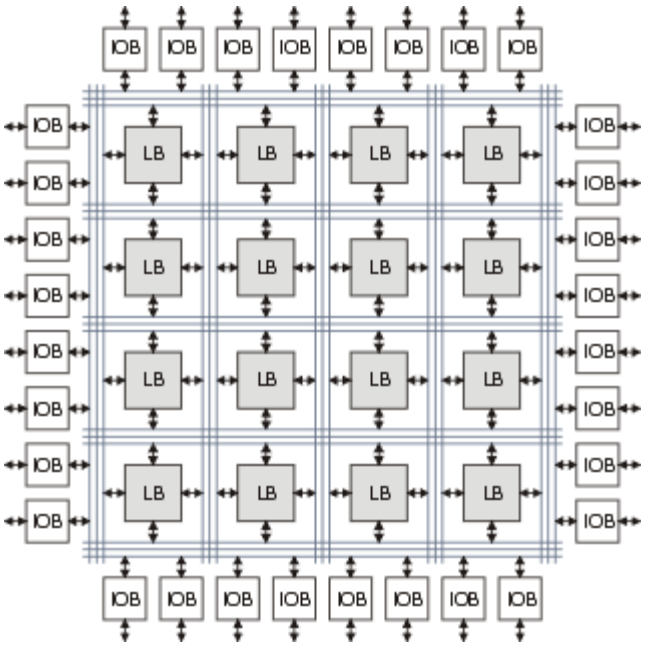
2 Teoretický rozbor

V této kapitole jsou uvedeny základní principy a pojmy z oblasti návrhu TSC obvodů v FPGA. Nejprve popíšeme strukturu obvodu FPGA. Dále se budeme věnovat bezpečnostním kódům a jejich vlastnostem a některým modelům poruch v logických obvodech a v hradlových polích. Nakonec uvedeme definici obvodů bezpečných proti poruchám a definici úplně samočinně kontrolovaných obvodů (TSC).

2.1 FPGA

Hradlová pole FPGA (Field programmable gate array) jsou programovatelné logické obvody. Mezi nejznámější výrobce FPGA patří firmy Xilinx, Altera a Lattice Semiconductor Corp.

Struktura FPGA je na obrázku 1. Obvod obsahuje pole konfigurovatelných logických bloků (jejich počet se pohybuje od několika set do řádově 100 000 bloků) vzájemně propojených programovatelným propojením. Propojení vnitřku obvodu s okolím zajišťují programovatelné vstupně- výstupní buňky umístěné po obvodu hradlového pole.



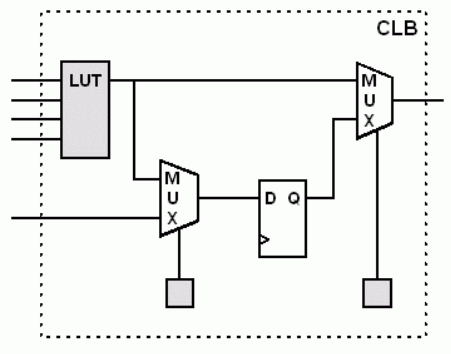
Obrázek 1 - struktura FPGA

Konfigurace FPGA, tj. nastavení logických bloků, vstupně-výstupních bloků a nastavení propojení, je nejčastěji uložena v paměti SRAM. Musí se proto po každém zapnutí do obvodu znovu nahrát z externího zdroje dat. Výhodou tohoto řešení je snadná rekonfigurace obvodu, buď celého, nebo po částech.

Konfigurovatelné logické bloky většiny současných FPGA jsou odvozeny ze základního zapojení podle obrázku 2. Obsahují čtyřvstupový funkční generátor, konfigurovatelné propojení a klopný obvod. Funkční generátor je realizován tabulkou 16 × 1 bit (označovanou také jako LUT). V LUT je

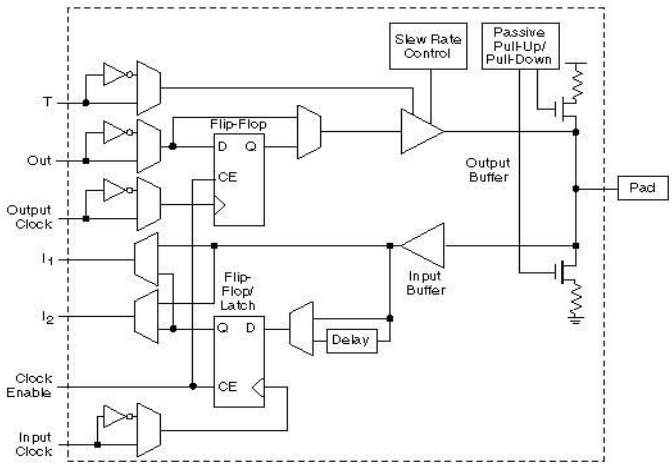
uložena úplná pravdivostní tabulka funkce generátoru. Pomocí LUT tak lze realizovat libovolnou funkci čtyř proměnných. Interní propojení je konfigurováno pomocí multiplexorů. Obsah LUT i nastavení multiplexorů je uloženo v konfigurační paměti obvodu.

Logické bloky bývají ve skutečnosti mnohem složitější. Například logický blok ve Virtex™-II od firmy Xilinx (tato firma nazývá logické bloky CLB – Configurable logic block) se skládá ze 4 řezů a každý z těchto řezů obsahuje dva funkční generátory, logiku pro urychlení přenosu, dva klopné obvody a řadu multiplexorů pro nastavení propojení uvnitř řezu. Každý funkční generátor může být zapojen také jako 16-bitová distribuovaná paměť RAM nebo 16-bitový posuvný registr.



Obrázek 2 - zjednodušené schéma konfigurovatelného logického bloku FPGA

Pro propojení vnitřku obvodu s piny obvodu obsahuje FPGA konfigurovatelné vstupně-výstupní buňky (IOB). Každá buňka může být nakonfigurována jako vstupní, výstupní nebo vstupně-výstupní, s klopným obvodem nebo bez něj. Buňka dále podporuje několik napěťových standardů, možnost připojení pull-up nebo pull-down rezistorů a obsahuje ochranu proti přepětí. Zjednodušené schéma této buňky je na obrázku 3.



Obrázek 3 - zjednodušené schéma vstupně-výstupní buňky FPGA

Další důležitou částí FPGA je propojení buněk. V FPGA je používáno několik typů propojení

- přímé propojení mezi sousedními CLB
- propojení mezi CLB pomocí propojovací matice (general routing matrix)
- long lines – horizontální a vertikální propojení pro přenos signálu na velkou vzdálenost s malým zpožděním

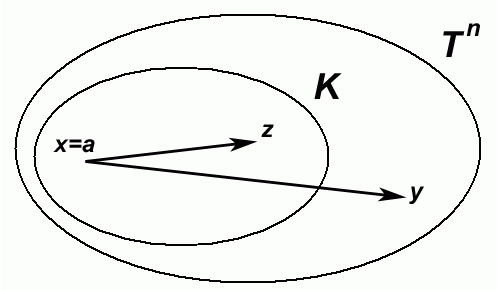
2.2 Bezpečnostní kódy

Bezpečnostní kódování je využíváno pro detekci nebo opravu chyby vzniklé průchodem informace reálným přenosovým kanálem. Této vlastnosti je dosaženo zavedením přídavné informace k přenášeným datům, viz [1].

Jedním ze základních rozdělení bezpečnostních kódů je rozdělení na detekční kódy (error-detection codes) a korekční, jinak taky samoopravné kódy (error-correction codes). Dále se budeme zabývat pouze kódy detekčními. Jak již bylo uvedeno výše, detekce chyb je možná díky redundanci. Princip detekce chyby je v [1] popsán takto:

Pokud pracujeme s blokovým kódem K na nějaké konečné abecedě T, potom množina všech slov délky n je $T^n = \{t_1 t_2 ... t_n \mid t_i \in T, i = 1,2,...,n\}$. Pokud délka všech slov v kódu K je rovna n, potom $K \subseteq T^n$ a slova množiny T^n dělíme na kódová slova (prvky množiny K) a nekódová slova (prvky množiny $T^n - K$). V reálném kanále jsou vysílána kódová slova a přijímána slova z množiny T^n. Pokud přijmeme nekódové slovo, říkáme, že jsme detekovali chybu. Pokud jsme přijali kódové slovo, pak je buď vše v pořádku, nebo jsme chybu nedetekovali.

Tři různé situace jsou zobrazeny na obrázku 4.



Obrázek 4 - Vliv kanálu na kódové slovo

Do kanálu bylo posláno kódové slovo *a*. Přijali jsme slova *x* nebo *y* nebo *z*. Původní slovo *a* samozřejmě neznáme a informaci o tom, zda je přijaté slovo shodné s vyslaným slovem by nám měl poskytnout detekční kód. Mohou tedy nastat tyto tři možnosti:

- Přijali jsme slovo *x*. Kontrolou kódu zjistíme, že *x* je kódové slovo, a proto jej správně považujeme za slovo shodné s původním slovem *a*.

- Přijali jsme slovo *y*. Kontrolou kódu zjistíme, že *y* není kódové slovo, a tím jsme detekovali chybu při přenosu. Bezpečnostní kód správně splnil svoji funkci.
- Přijali jsme slovo *z*. Kontrolou kódu zjistíme, že *z* je kódové slovo, a proto jej nesprávně považujeme za slovo shodné s původním slovem *a*. V tomto případě použitý bezpečnostní kód selhal.

Přenosové kanály se liší charakterem (typem) chyb, které v nich mohou nastat a jednotlivé kódy se liší počtem a typem chyb, které jsou schopny detekovat. Pro efektivní využití bezpečnostního kódu je vhodné volit kód s ohledem na vlastnosti kanálu.

Kanály (pro binární slova) můžeme podle [2] rozdělit na:

- Symetrické kanály – kanály, u kterých je pravděpodobnost změny bitu z hodnoty logická 0 na logická 1 stejná jako pravděpodobnost změny z hodnoty logická 1 na logická 0.
- Nesymetrické kanály – kanály, ve kterých všechny chyby obsahují pouze změny z log. 1 na log. 0, případně změny z log. 0 na log. 1. Směr změny je předem znám.
- Kanály s jednosměrnými chybami (unidirectional errors) – kanály, u kterých se mohou objevit oba typy chyb, ale v každém přijatém slově se mohou vyskytnout pouze chyby jednoho typu. Příklad slov změněných kanálem s touto vlastností je v tabulce 1.

Tabulka 1- příklad jednosměrných chyb

a	***y***
110010	100000
110010	110110

Počet chyb, které je kód schopen detekovat u symetrického kanálu je určen kódovou vzdáleností. Kód s kódovou vzdáleností d je schopen detekovat až $d-1$ chyb v přijatém slově.

V případě kanálů s jednosměrnými chybami existuje podle [2] podmínka, při jejímž splnění je kód schopen detekovat všechny jednosměrné chyby:

Nechť $N(x,y)$ je počet znaků, které mají ve slově x hodnotu logická 1 a ve slově y hodnotu logická 0. Kód K je schopen detekovat všechny jednosměrné poruchy, jestliže pro všechna kódová slova $x, y \in K$ platí $N(x,y) \geq 1$ a zároveň $N(y,x) \geq 1$.

Dalším rozdělením bezpečnostních kódů je rozdělení na kódy systematické a kódy nesystematické. Definice systematického kódu je uvedena v [1]:

Blokový kód K, $K \in T^n$ se nazývá systematický, jestliže existuje číslo $k < n$ takové, že pro každé slovo $v_1 v_2 \ldots v_k \in T^k$ existuje právě jedno kódové slovo $v_1 v_2 \ldots v_k v_{k+1} \ldots v_n$. Kódování $\varphi : T^k \to K$ definované předpisem $\varphi(v_1 v_2 \ldots v_k) = v_1 v_2 \ldots v_k v_{k+1} \ldots v_n$ se nazývá systematické.

U systematických i nesystematických kódů se tedy provádí transformace původních slov na zakódovaná slova, ale liší se v tom, jestli je původní slovo přímo součástí zakódovaného slova, nebo jestli lze původní slovo získat pouze opačnou transformací. Tato vlastnost má vliv na možnosti použití kódu.

V následujících odstavcích si podrobněji popíšeme kódy používané při návrhu úplně samočinně kontrolovaných obvodů (TSC).

2.2.1 Bezpečnostní kódy systematické

Sudá parita (even parity)

Parita je jedním z nejznámějších a nejčastěji používaných bezpečnostních kódů. Jeho výhodou je velmi malá redundance - k zabezpečení informace je použit jeden kontrolní bit. Jeho hodnota je taková, aby celkový počet bitů s hodnotou log. 1 byl sudý. Kódová vzdálenost tohoto kódu je rovna 2, proto je schopný detekovat pouze jednu chybu (resp. lichý počet chyb). Není schopen detekovat jednosměrné chyby. Příklad tohoto kódu je uveden v tabulce 2.

Bergerův kód (Berger code)

Další ze systematických kódů je Bergerův kód. Kódová vzdálenost tohoto kódu je také rovna dvěma. Pro k informačních bitů je potřeba $m = \lceil \log_2(k+1) \rceil$ kontrolních bitů. V kontrolních bitech je uložen binárně vyjádřený buď počet bitů informační části s hodnotou logická 0 (kód B_0), nebo počet bitů s hodnotou logická 1 (kód B_1). Kód B_0 je pro účely návrhu TSC obvodů výhodnější, protože je schopný detekovat všechny jednosměrné chyby – jedná se dokonce o optimální kód schopný detekovat tyto chyby (viz [2]). V tabulce 2 je uveden příklad tohoto kódu.

Tabulka 2 - Příklady systematických kódů

kódovaná data	***sudá parita***	***Bergerův kód B_0***
000	000**0**	000**11**
001	001**1**	001**10**
010	010**1**	010**10**
011	011**0**	011**01**
100	100**1**	100**10**
101	101**0**	101**01**
110	110**0**	110**01**
111	111**1**	111**00**

2.2.2 Bezpečnostní kódy nesystematické

Kód M z N

Kódy M z N se vyznačují tím, že každé kódové slovo obsahuje m bitů s hodnotou logická 1. Tyto kódy se používají například pro kódování vnitřních stavů automatu, výstupů řadičů a podobně. Kódem *M z N* lze zakódovat maximálně $\binom{n}{m}$ různých kombinací. Nejčastěji používané varianty jsou kód *1 z N* a kód *k z 2k*. Druhý uváděný umožňuje zakódovat největší počet kombinací při dané délce slova, další jeho výhodou je snadný návrh TSC hlídače kódu (viz [3]). Tyto kódy mají kódovou vzdálenost rovnu dvěma a jsou schopny detekovat všechny jednosměrné chyby.

2.3 Poruchy v logických obvodech

Při testování poruch nebo návrhu obvodů bezpečných proti poruchám je nutné používat místo skutečných fyzikálních poruch (defects) modely poruch (fault model). Tyto modely (nazývané také jako logické poruchy) by měly co nejvíce pokrývat fyzikální poruchy, které mohou v obvodu implementovaném určitou technologií nastat. Následující výčet obsahuje některé nejznámější poruchy.

2.3.1 Poruchy typu t (stuck-at faults)

Diagnostika číslicových obvodů dosud vystačila s malým počtem logických poruch. Nejznámější a nejstarší jsou poruchy typu t (trvalé poruchy).

- trvalá nula (t_0) – porucha, která se projevuje konstantní hodnotou logická 0 na vodiči.
- trvalá jednička (t_1) – porucha, která se projevuje konstantní hodnotou logická 1 na vodiči.

2.3.2 Zkraty (bridging faults)

Zkraty vznikají propojením vodičů, které by za normálních okolností nebyly propojeny. Výsledná hodnota závisí na použité technologii. Tyto poruchy lze modelovat pomocí fiktivního logického členu zapojeného mezi oba zasažené signály.

2.3.3 Přechodné poruchy

Tyto poruchy jsou způsobeny dopadem vysoce nabité částice na polovodičový prvek. Je-li energie takové částice dostatečně velká, může změnit výstup zasažené součástky. Tato událost je označována jako SE (Single Event) nebo také SEP (Single Event Phenomena). Poruchy tohoto typu lze podle jejich následků rozdělit na:

- Single Event Upset (SEU) – SE způsobí změnu informace uložené v klopném obvodu
- Single Event Transient (SET) – SE dočasně změní výstup některého z hradel v obvodu a způsobí tak pulz na některém ze signálů v obvodu.

2.4 Poruchy v FPGA

Nejčastěji používaná hradlová pole FPGA mají konfiguraci uloženou v paměti SRAM. Jejich výhodou je snadná změna konfigurace, a tím i funkce obvodu. Nevýhodou je skutečnost, že výskyt SEU může výrazně ovlivnit funkci obvodu. Mohou nastat dvě situace:

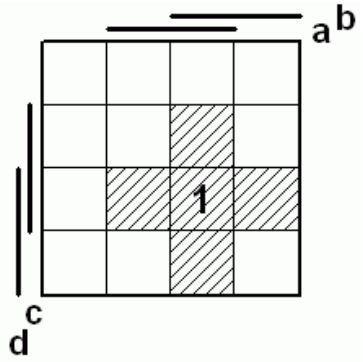
- Porucha SEU změní obsah paměťového prvku, který je součástí navrženého obvodu. Může například změnit obsah datového registru, stavového registru a podobně. Podobný případ může nastat tehdy, jestliže je pulz způsobený poruchou SET přiveden kombinační logikou na vstupy klopného obvodu v okamžiku hodinového taktu nebo pokud SET přímo ovlivní synchronní nebo asynchronní řídící signály klopného obvodu.
- Porucha SEU změní obsah paměti uchovávající konfiguraci obvodu. Může tak být změněn obsah LUT, a tím i funkce CLB, propojení uvnitř CLB i propojení mezi CLB.

Dále se budeme zabývat pouze poruchami typu SEU, které nezmění propojení v obvodu. Tzn. zaměříme se pouze na ty poruchy, které způsobují změnu obsahu paměťového prvku, který je součástí obvodu nebo změnu funkce LUT.

Změna obsahu synchronního paměťového prvku způsobená SE je dočasná porucha, která se projeví chybným výstupem prvku pouze po dobu jednoho taktu. Její účinky lze pro každý krok testu simulovat poruchou typu t opačné hodnoty, než je aktuální hodnota daného paměťového prvku.

Složitější případ nastane při změně konfigurační paměti LUT následkem poruchy SEU. Tato změna je na rozdíl od předchozího případu trvalá. Možnosti jejího otestování jsou ale odlišné od testování poruch typu t. Jako příklad uvažujme čtyřvstupé hradlo AND realizované pomocí LUT. Minimální test pro poruchy typu t na vstupech a výstupu LUT je na obrázku 5.

abcd	a	b	c	d	y
1111	t_0	t_0	t_0	t_0	t_0
1110				t_1	t_1
1101			t_1		t_1
1011		t_1			t_1
0111	t_1				t_1



Obrázek 5 - Minimální test pro poruchy typu t a Karnaughova mapa pro AND4

Funkce LUT je ale realizována tabulkou. Každému poli tabulky odpovídá jeden bit konfigurační paměti a libovolný z těchto bitů může být ovlivněn poruchou SEU. Karnaughova mapa na obrázku 5 odpovídá tabulce LUT pro čtyřvstupový logický součin. Šrafovaně jsou vyznačená pole, která byla otestována minimálním testem pro poruchy typu t. Je zřejmé, že testy navržené pro poruchy typu t nestačí k otestování všech poruch SEU v konfigurační paměti LUT a je nutné použít triviální test.

V některých dokumentech zabývajících se návrhem samočinně kontrolovaných obvodů (definice těchto obvodů je uvedena v kapitole 2.6) jsou poruchy SEU v konfigurační paměti LUT (označované také jako single functional fault) nahrazovány poruchami typu t na výstupu LUT. Toto nahrazení je možné provést pouze pro ověření, zdali je obvod bezpečný proti poruchám, ale nelze jej použít pro ověření samočinné kontroly obvodu. Toto tvrzení dokážeme v následujících odstavcích.

V každém okamžiku je hodnotami na vstupních pinech LUT vybrán jeden bit konfigurační paměti LUT a výstup LUT odpovídá hodnotě tohoto bitu. Změnu způsobenou poruchou SEU můžeme nahradit poruchou typu t na výstupu LUT s opačnou hodnotou, než byla původní hodnota bitu. V daném okamžiku jsou obě poruchy ekvivalentní. Vlivem omezené množiny vstupních slov obvodu a vlivem redundance v obvodu mohou mít některé LUT omezenou množinu vstupních slov. Poruchy SEU v bitech odpovídajících slovům, které se na vstupu LUT nemohou objevit, se nikdy neprojeví, a proto nemají z hlediska bezpečnosti proti poruchám význam. Proto je obvod bezpečný proti všem poruchám typu t na výstupních pinech LUT bezpečný i proti všem poruchám typu SEU v konfigurační paměti LUT.

Poruchy SEU v konfigurační paměti LUT ale nelze nahrazovat poruchami typu t při ověřování samočinné kontroly obvodu. Pro otestování všech poruch SEU by bylo nutné provést triviální test, zatímco pro poruchy typu t na výstupech LUT to nutné není.

2.5 Obvody bezpečné proti poruchám

Jako obvody bezpečné proti poruchám (Fault secure – FS) označujeme takové obvody, které za běhu kontrolují svoji funkci, a tím jsou schopny detekovat přítomnost trvalé i dočasné poruchy. Hlavní účel on-line detekce (concurrent error detection – CED) je detekce poruch, které mohou nastat při běhu systému – nenahrazuje tedy vlastní verifikaci a výrobní testy obvodu. Přesná definice obvodů bezpečných proti poruchám je uvedena například v [3]:

> Předpokládejme, že obvod realizuje logickou funkci $z = Z(x)$, kde x je vstupní a z výstupní vektor. Tento obvod budeme dále označovat jako obvod Z. Množinu vstupních vektorů označíme $\boldsymbol{X}$, množinu výstupních vektorů $\boldsymbol{Z}$. Výstup obvodu Z je bezpečnostně kódován, takže platí $\boldsymbol{Z} \subseteq B^n$, kde n je počet bitů vektoru z a B^n je množina všech vektorů délky n. Označme množinu poruch, proti nimž má být obvod zabezpečený, symbolem $\boldsymbol{F}$. Pokud se v obvodu Z vyskytne porucha $f \in \boldsymbol{F}$, bude tento obvod realizovat funkci $Z(\boldsymbol{X}, f)$, která se obecně může lišit od funkce $Z(\boldsymbol{X})$.
>
> Obvod Z je pro množinu vstupních vektorů $\boldsymbol{X}$ bezpečný proti poruchám z množiny $\boldsymbol{F}$, jestliže pro každý vstupní vektor $x \in \boldsymbol{X}$ a každou poruchu $f \in \boldsymbol{F}$ platí buď $Z(\boldsymbol{X}, f) \notin \boldsymbol{Z}$, nebo $Z(\boldsymbol{X}, f) = Z(\boldsymbol{X})$.

Tuto definici lze vysvětlit tak, že pro uvažovanou množinu poruch nemůže nastat situace, ve které by se vlivem poruchy v obvodu pro libovolné povolené vstupní slovo objevila na výstupu obvodu nedetekovatelná chyba. Porucha se buď neprojeví vůbec, nebo se projeví tak, že ji lze detekovat.

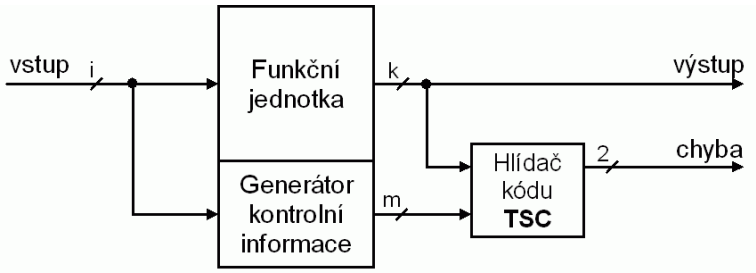
Existuje několik metod pro zabezpečení obvodu proti poruchám:

- Zdvojení - nejjednodušší metoda návrhu obvodu bezpečného proti poruchám. Obvod obsahuje dvě shodné funkční jednotky, jejichž výstupy jsou vzájemně porovnávány a při detekci rozdílných hodnot je indikována chyba. Nevýhodou je více než stoprocentní redundance. Podrobnější informace o této metodě lze najít například v [4].

- Kontrola inverzní funkcí – u této metody je k funkční jednotce přidána jednotka s inverzní funkcí. Jejím vstupem je výstup první jednotky a její výstupy se porovnávají s původními vstupy obvodu. Při odlišných hodnotách je opět indikována chyba. Výhodou této metody je vyšší odolnost proti poruchám oproti předchozí metodě, nevýhodou je opět vysoká redundance obvodu a také fakt, že metodu lze použít jen tehdy, pokud inverzní funkce existuje.

- Použití bezpečnostního kódu – výstupy obvodu jsou zakódovány některým z bezpečnostních systematických kódů. Informační část i kontrolní část kódu jsou ve většině případů počítány samostatně, výsledný kód je kontrolován hlídačem kódu.

Dále se budeme zabývat pouze poslední metodou uvedenou v předchozím přehledu, tedy metodou používající bezpečnostní kódování. Princip této metody je zobrazen na obrázku 6. Kromě výpočtu vlastní funkce je ještě prováděn výpočet hodnoty kontrolních bitů a výsledky z obou částí jsou přivedeny do hlídače kódu.



Obrázek 6 - použití bezpečnostního kódu pro on-line detekci chyb

Z principu funkce těchto obvodů dále vyplývá, že spolehlivá detekce poruch je závislá na správné funkci hlídače kódu. Ty se proto navrhují jako úplně samočinně kontrolované obvody, které jsou popsány v kapitole 2.6.

2.6 Úplně samočinně kontrolované obvody

Při použití obvodů bezpečných proti poruchám je podle definice zajištěno, že se žádná porucha neprojeví nesprávným kódovým slovem na výstupu obvodu. Není ale zaručeno, že pro každou poruchu z uvažované množiny poruch existuje takový vstupní vektor, který by způsobil její projevení nekódovým slovem na výstupu. Tento fakt sice nemá vliv na správnou funkci obvodu, ale znemožňuje zjištění skutečného stavu obvodu, tedy jestli se v něm nenachází nějaká porucha. Porucha, která se neprojeví, může později v kombinaci s další poruchou způsobit nedetekovatelnou chybu na výstupu. Aby se tomu zabránilo, musel by se tento obvod podrobovat pravidelným diagnostickým testům. To v některých případech buď není možné, nebo by nutná úprava neúměrně zvýšila složitost návrhu obvodu. Řešením je použití úplně samočinně kontrolovaného obvodu.

Pro definici úplně samočinně kontrolovaného obvodu je nutné ještě uvést definici samočinně testovaného obvodu (self testing – ST) [3]:

Obvod Z je samočinně testovaný pro množinu vstupních vektorů $\boldsymbol{X}$ a množinu poruch $\boldsymbol{F}$, jestliže pro každou poruchu $f \in \boldsymbol{F}$ existuje vstupní vektor $x \in \boldsymbol{X}$ takový, že platí $Z(x, f) \notin \boldsymbol{Z}$.

Úplně samočinně kontrolovaný obvod (totally self-checking – TSC) je takový obvod, který je bezpečný proti poruchám a současně samočinně testovaný.

Samočinné testování zajišťuje, že se každá porucha v obvodu po určitém čase projeví detekovatelnou chybou na výstupu obvodu. Pokud je doba mezi výskytem poruch dostatečně dlouhá na to, aby se na vstupech obvodu stačily vystřídat všechny vektory tvořící úplný test, je zajištěno, že se první porucha v obvodu projeví dříve, než další poruchy v obvodu způsobí nedetekovatelnou chybu na výstupu.

Takové obvody potom nepotřebují periodickou diagnostiku, a to je hlavní význam TSC obvodů.

2.6.1 Úplně samočinně kontrolovaný hlídač kódu

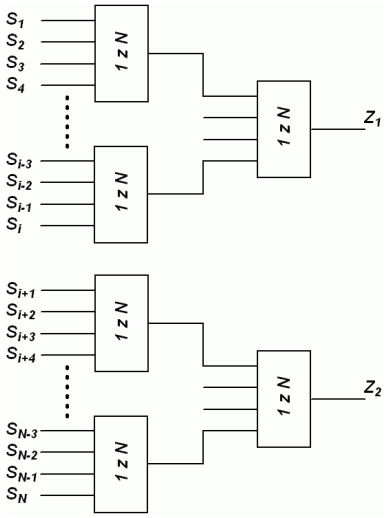
Pokud se úplně samočinně kontrolovaný obvod použije jako hlídač kódu, mívá tento obvod dva výstupy indikující správné nebo nesprávné kódové slovo na vstupu hlídače. Tyto výstupy jsou zakódované kódem 1 ze 2. Jestliže se na vstupu hlídače vyskytuje kódové slovo, je i na výstupu hlídače kódové slovo (pouze jeden výstupní signál má hodnotu log. 1). V ostatních případech bude na výstupu nekódové slovo. Souhrnná reakce TSC hlídače kódu je uvedena v tabulce 3.

Tabulka 3- funkce TSC hlídače kódu

Výstup TSC hlídače kódu		***Výsledek kontroly kódu***
0	0	nekódové slovo
0	1	kódové slovo
1	0	kódové slovo
1	1	nekódové slovo

Dva signály jsou nutné z důvodu splnění podmínek definujících úplně samočinně kontrolované obvody. V případě jediného výstupního signálu by nebylo možné odlišit chybový stav od bezchybového stavu.

Je nutné si uvědomit, že TSC obvody jsou navrhovány pro určitou množinu poruch a pro jinou množinu poruch nemusí mít tuto vlastnost. Jako příklad si uvedeme hlídač kódu 1 z N z obrázku 7.



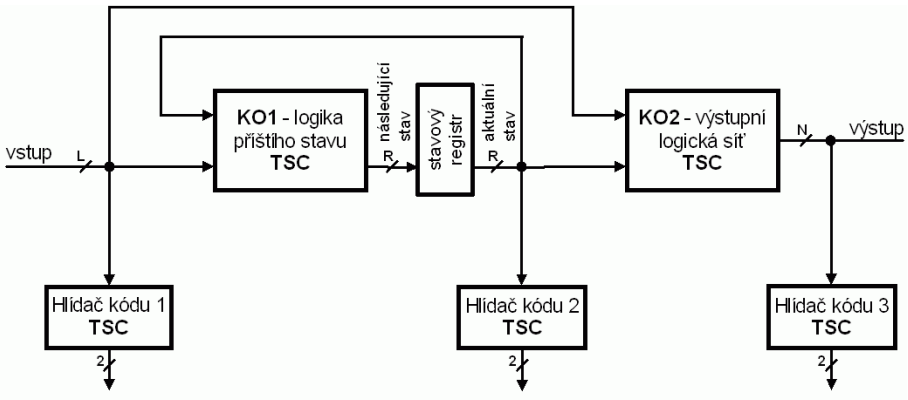
Obrázek 7 - Hlídač kódu 1 z N

Obvod je implementován v FPGA, každý funkční blok představuje jednu LUT. Funkční blok má na výstupu hodnotu logická 1 pouze tehdy, když má právě jeden jeho vstupní pin hodnotu logická 1. Na vstupu každého bloku se vyskytují slova z množiny {0000, 0001, 0010, 0100, 1000}. Lze snadno dokázat, že pro poruchy typu t na vstupních a výstupních pinech LUT je tento obvod úplně samočinně kontrolovaný. Pokud bychom ale uvažovali poruchy typu SEU v konfigurační paměti LUT, bude tento hlídač pouze bezpečný proti poruchám, ale nebude samočinně testovaný, a proto nebude ani úplně samočinně kontrolovaný.

2.6.2 Úplně samočinně kontrolované sekvenční obvody

U úplně samočinně kontrolovaných sekvenčních obvodů je nutné zajistit, aby pro libovolnou posloupnost slov na vstupu obvodu odpovídala výstupní posloupnost požadované funkci obvodu. V opačném případě musí obvod signalizovat chybu. Protože výstupní posloupnost může být ovlivněna jak poruchou v části pro výpočet následujícího vnitřního stavu obvodu, tak poruchou v části pro výpočet výstupu obvodu, musí být kontrolována správná funkce obou těchto částí. Do množiny poruch se běžně nepočítají poruchy na hodinovém signálu. V [5] je uveden přehled metod používaných pro návrh TSC sekvenčních obvodů.

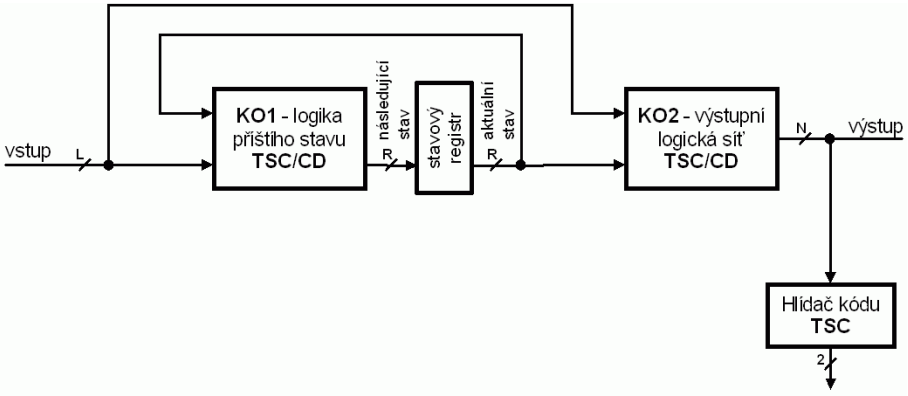
První metoda (viz [5]) využívá zakódování vstupů, výstupů a vnitřního stavu obvodu některým z bezpečnostních kódů a trojici samostatných TSC hlídačů kódu, po jednom pro vstup, výstup a vnitřní stav automatu. Kombinační obvody pro výpočet následujícího stavu a výstupu obvodu jsou také úplně samočinně kontrolované. Princip této metody je na obrázku 8. V některých případech není používáno zakódování vstupů obvodu. Pak samozřejmě není možné detekovat poruchy na primárních vstupech obvodu. Varianta se zakódovaným vstupem se používá například při návrhu sítě automatů, kdy je v celé síti použit jeden bezpečnostní kód a lze tak přímo propojit komunikující automaty.



Obrázek 8 - TSC sekvenční obvod - varianta 1

V [6] je uveden návrh TSC sekvenčního obvodu s použitím liché parity. Později se budeme zabývat návrhem TSC obvodů využívajícího kódy M z N a implementaci bez použití invertorů.

Snaha o zmenšení redundance vedla k druhé metodě, používající pouze jeden hlídač kódu na výstupu obvodu a funkční jednotky schopné propagovat chyby ze svých vstupů na svoje výstupy. Obě funkční jednotky tedy kromě toho, že jsou úplně samočinně kontrolované, musí splňovat i druhou podmínku, a to že na jejich výstupu se může objevit kódové slovo pouze tehdy, je-li kódové slovo i na jejich vstupu (tato podmínka je označována jako Code-disjoint). Princip funkce je na obrázku9.



Obrázek 9 - TSC sekvenční obvod - varianta 2

3 Předchozí práce

V této kapitole budou představeny dva způsoby návrhu TSC obvodů. První způsob je označován jako MD architektura a je vhodný pro automaty, které mají malý počet různých výstupních slov. Hlídač obvodu netestuje, jestli je na výstupu kódové slovo, ale jestli je na výstupu správné výstupní slovo. Obvod vnitřně pracuje v kódu 1 z N a společně se způsobem implementace je tak zajištěno, že všechny uvažované poruchy s výjimkou poruch na vstupních pinech obvodu budou detekovány.

V druhé metodě je použit kód schopný detekovat jednosměrné chyby pro zabezpečení vstupu, výstupu i stavových signálů obvodu. Obvod je navržen tak, aby se každá porucha projevila jednosměrnou chybou, a tak je vzhledem k použitému kódu zajištěna bezpečnost obvodu proti poruchám.

3.1 MD architektura

Způsob návrhu TSC obvodů označovaný jako MD architektura byl popsán v [7] a dále se jím zabývají v [8], [9], [10] a [11].

MD architektura je vhodná pro implementaci automatů, které mají tyto vlastnosti:

- mají úplně definovanou přechodovou funkci (tzn. neexistuje v nich stav, který by měl nedefinovanou přechodovou funkci pro nějaké vstupní slovo)
- počet různých výstupních slov bývá výrazně menší než 2^n , kde *n* je počet bitů výstupu
- přechody z každého stavu jsou ovlivněny jen několika signály vstupu

Tyto vlastnosti umožňují podle autorů dosáhnout menší velikosti úplně samočinně kontrolovaného obvodu, než když je použito bezpečnostního kódování nebo dvoudrátová logika. První podmínka je nutná pro zaručení úplné samočinné kontroly obvodu, další dvě vlastnosti mají vliv na výslednou velikost obvodu.

Obvod podle této architektury by měl být bezpečný proti poruchám t_0 a t_1 na vstupních a výstupních pinech CLB a proti poruchám způsobujícím jednu funkční chybu LUT. Poruchy na vstupu obvodu jsou nedetekovatelné. Toto tvrzení ověříme v kapitole 3.1.3.

3.1.1 Struktura obvodu

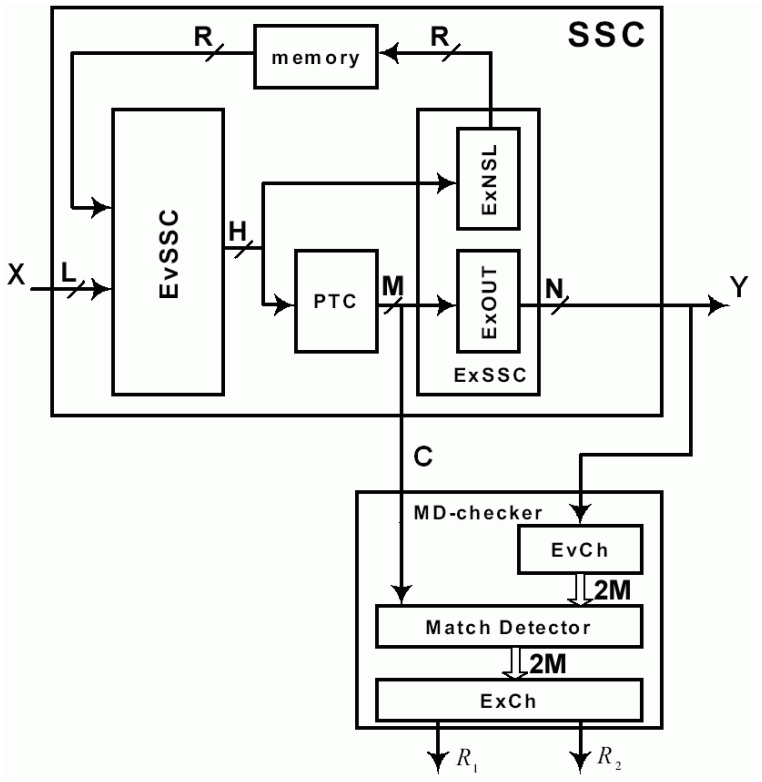
Obvod podle MD architektury se skládá ze dvou celků, hlídaného sekvenčního obvodu a hlídače. Vstup ani výstup obvodu není zakódován bezpečnostním kódem. Hlídač kontroluje, jestli v obvodu nenastala nějaká porucha a jestli je na výstupu sekvenčního obvodu správné výstupní slovo.

Schéma zapojení je na obrázku 10. Konstanty použité pro popis sekvenčního obvodu jsou v tabulce 4. Jak již bylo uvedeno výše, obvod se skládá ze dvou celků, ve schématu označených jako SSC a MD-checker.

- SSC – kontrolovaný sekvenční obvod. Komunikace mezi všemi vnitřními jednotkami probíhá v kódu 1 z N. Způsob implementace zaručuje, že se každá porucha projeví buď na kontrolním výstupu C a/nebo na původním výstupu Y. Na kontrolním výstupu C je v kódu 1 z N (konkrétně 1 z M) index slova, které bude v případě bezchybné funkce na výstupu Y.
- MD-checker – úplně samočinně kontrolovaný hlídač. Nepracuje na principu kontroly kódu na výstupu Y hlídaného sekvenčního obvodu, ale vyhodnocuje, zdali je na něm správné slovo. Hodnota z výstupu Y je převedena zpět do kódu 1 z N (1 z M) a tato hodnota je porovnána s hodnotou z kontrolního výstupu C sekvenčního obvodu. Podle výsledku porovnání je pak nastaven výstup hlídače. Pro komunikaci mezi jednotkami celku MD-checker se používá kód 1 z N v dvoudrátové variantě

Tabulka 4 - číselné charakteristiky obvodu podle MD-architektury

označení	*význam*
L	šířka vstupu (počet signálů)
N	šířka výstupu (počet signálů)
H	celkový počet přechodů automatu (počet termů)
M	počet různých výstupních slov
R	počet stavů



Obrázek 10 - MD architektura

Sekvenční obvod se skládá ze tří jednotek: EvSSC (Evolution Block of Synchronous Sequential Circuit), PTC (Product Term Compressor) a ExSSC (Execution block of Synchronous Sequential Circuit). ExSSC se skládá z dalších dvou částí: ExOUT a ExNSL.

EvSSC

Vstupem jednotky EvSSC jsou primární vstup obvodu a výstup z paměťových elementů. Výstupní signály EvSSC odpovídají jednotlivým přechodům automatu. V libovolném okamžiku má právě jeden výstupní signál hodnotu logická 1, výstup tak odpovídá kódu 1 z N (konkrétně 1 z H).

Každý výstup odpovídá jednomu přechodu, a proto je vyhodnocen jako logický součin vstupních signálů (které jej ovlivňují) a jednoho signálu z klopných obvodů odpovídajícího aktuálnímu vnitřnímu stavu pro daný přechod (vnitřní stav je zakódován kódem 1 z N). V této jednotce není povoleno sdílení prostředků.

PTC

Na vstupu jednotky PTC je informace o aktivním přechodu automatu v kódu 1 z H (v případě bezporuchového provozu má právě jeden signál hodnotu log. 1), na výstupu PTC je index odpovídajícího výstupního slova sekvenčního obvodu v kódu 1 z M. Jednotka také slouží jako hlídač kódu na výstupu jednotky EvSSC.

Jednotka redukuje kód 1 z H na kód 1 z M, $H \geq M$. Pro implementaci se používá první varianta buňky O-Cell z obrázku 12, každému výstupnímu signálu odpovídá stromové propojení těchto buněk. Z principu funkce této jednotky vyplývá, že opět nedochází k žádnému sdílení prostředků.

ExOUT

Jednotka ExOUT je určena pro výpočet výstupních slov sekvenčního obvodu. Na jejím vstupu je v kódu 1 z M index výstupního slova, na výstupu jednotky je odpovídající slovo.

Pro implementaci se používá logický součet. Každý výstupní signál je vyhodnocen jako logický součet vstupních signálů odpovídajících slovům, ve kterých má mít daný signál hodnotu logická 1.

ExNSL

Jednotka ExNSL pro výpočet následujícího stavu. Na vstupu jednotky ExNSL je informace o aktivním přechodu v kódu 1 z H, na výstupu je hodnota následujícího stavu v kódu 1 z R.

Pro implementaci se používá logický součet, stejně jako u jednotky ExOUT.

Úplně samočinně kontrolovaný hlídač, označený jako MD-checker se také skládá ze tří částí: EvCh (Evolution Block of Checker), MD (Match detector) a ExCh (Execution Block of Checker).

EvCh

Jednotka EvCh převádí výstupní slova sekvenčního obvodu zpět do kódu 1 z M, tentokrát ale v dvouvodičovém vyjádření. Každá dvojice odpovídá právě jednomu povolenému výstupnímu slovu. V případě správné funkce sekvenčního obvodu a hlídače má právě jedna výstupní dvojice (S_i, V_i) hodnotu (1,0), všechny ostatní dvojice mají hodnotu (0,1).

Jednotka je implementována jako strom, resp. les bloků A-Cell podle obrázku 13.

MD

Jednotka porovnává původní kód 1 z M z kontrolního výstupu C sekvenčního obvodu s dvoudrátovým vyjádřením kódu 1 z M z jednotky EvCh. Slouží také jako hlídač kódu na kontrolním výstupu C.

Funkce pro porovnání jednoho páru PTC_i a (S_i, V_i) je uvedena v tabulce 5. V případě, že se v obvodu nevyskytla porucha, je na výstupu MD stejná hodnota jako na výstupu EvCh.

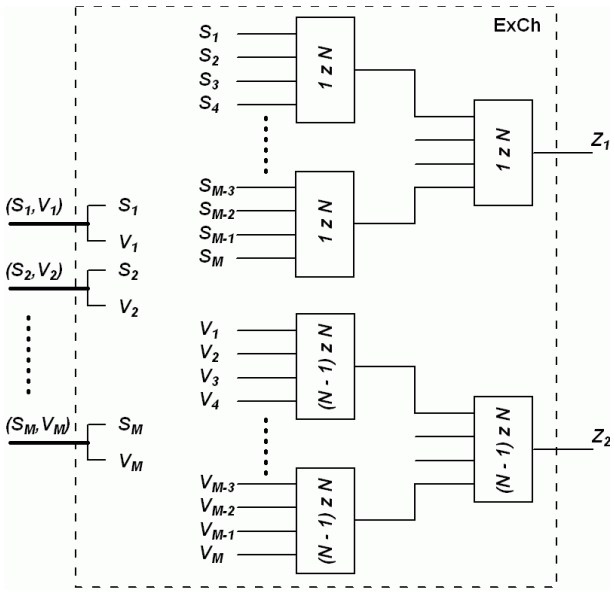
Tabulka 5 - funkce MD detektoru v MD architektuře

PTC_I	S_i	V_i	S_O	V_O
1	1	0	1	0
0	0	1	0	1
1	0	1	1	1
0	1	0	0	0
-	0	0	0	0
-	1	1	1	1

ExCh

Jednotka vyhodnocující výsledky z MD. Na jejím vstupu je M hodnot v dvoudrátové logice, na jejím výstupu je klasická dvoudrátová signalizace poruchy v obvodu. Obvod signalizuje bezporuchový stav jedině v případě, že právě jedna z dvojic (S_i, V_i) má hodnotu (1,0) a všechny ostatní mají hodnotu (0,1).

Celá jednotka ExCh je složena z buněk O-Cell (obrázek 12). První polovina obvodu se skládá z prvního typu buněk a kontroluje, jestli v signálech S_i ze všech dvojic (S_i, V_i) má právě jeden hodnotu log.1. Druhá polovina se skládá z buněk O-Cell druhého typu a kontroluje, jestli v signálech V_i ze všech dvojic (S_i, V_i) má právě jeden hodnotu log.0.

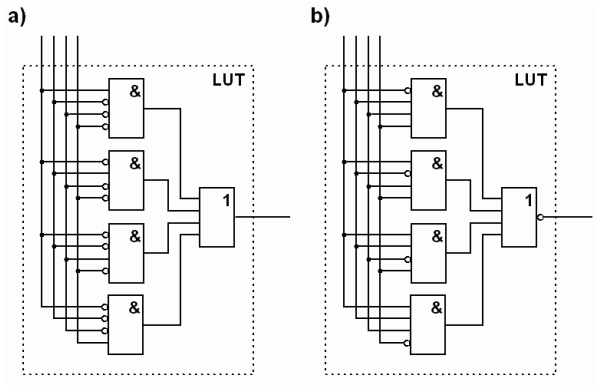


Obrázek 11 - Jednotka ExCh v MD architektuře

Při implementaci celého obvodu se, kromě jiného, používají buňky dvojího typu, O-Cell a A-Cell.

O-Cell

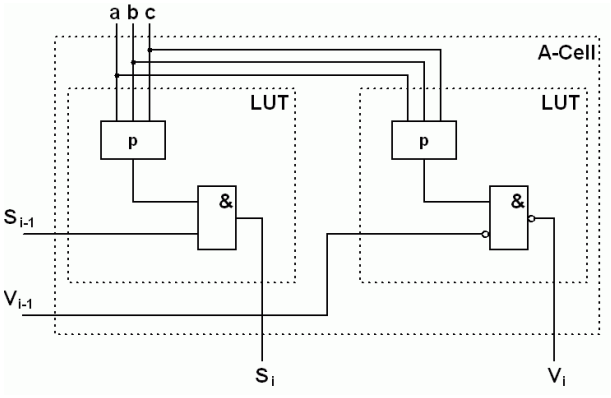
Buňka implementující buď funkci 1 z N nebo funkci (N-1) z N. U prvního typu (na obrázku 12 označeného jako a) je na výstupu log. 1 pouze tehdy, jestliže má právě jeden vstupní signál hodnotu logická 1. U druhého typu (na obrázku 12 označeného jako b) je na výstupu log. 0 pouze tehdy, jestliže má právě jeden vstupní signál hodnotu log. 0. Buňka je tvořena jednou LUT.



Obrázek 12 - O-Cell buňka MD architektury

A-Cell

Buňka pro kaskádní výpočet „dlouhého“ součinu. Princip zapojení je na obrázku 13. Funkce *p* je v obou částech buňky stejná a implementuje některý term proměnných *a*, *b* a *c*. Pro kaskádní vstup a výstup je použita dvoudrátová logika. Výsledky předchozí části součinu jsou přivedeny na vstupy S_{i-1} a V_{i-1}, výsledek součinu je na výstupech S_i a V_i.



Obrázek 13 - A-Cell

3.1.2 Funkce obvodu

Jednotka EvSSC aktivuje (nastaví na hodnotu log. 1) na svém výstupu signál odpovídající kombinaci vstupních signálů a aktuálního vnitřního stavu. Automat je úplně definovaný, proto má v každém okamžiku právě jeden signál hodnotu log. 1. Podle výstupu EvSSC připraví jednotka ExNSL hodnotu následujícího stavu a jednotka PTC index výstupního slova. Z indexu výstupního slova je v ExOUT vypočítáno skutečné výstupní slovo. Výstupní slovo a index výstupního slova je přiveden do hlídače.

Hlídač převede v EvCh výstupní slovo automatu zpět na index výstupního slova a ten je pak v MD porovnán s informací z kontrolního výstupu C sekvenčního obvodu. ExCh podle výsledku porovnání nastaví výstup hlídače.

3.1.3 Ověření úplně samočinné kontroly

Autoři MD-architektury si jako model poruch zvolili poruchy typu t na vstupních a výstupních pinech CLB a poruchy způsobující jednu funkční chybu LUT (SEU v konfigurační paměti LUT). Neuvažují poruchy na primárních vstupech obvodu. Pro ověření úplné samočinné kontroly nahrazují poruchy SEU v konfigurační paměti LUT poruchami typu t na výstupu LUT. V kapitole 2.4 jsme dokázali, že takto lze postupovat pouze při ověřování bezpečnosti obvodu proti poruchám, ale ne při ověřování samočinné kontroly obvodu. Obvod tedy nemůže být pro požadovanou množinu poruch samočinně kontrolovaný.

Jiná situace nastane, pokud budeme uvažovat pouze poruchy typu t na vstupech a výstupu CLB. V následujících odstavcích ověříme bezpečnost jednotlivých jednotek MD architektury proti poruchám typu t na vstupech a výstupech CLB a schopnost samočinné kontroly.

EvSSC

Při implementaci jednotky EvSSC není povoleno sdílení prostředků mezi jednotlivými termy. Jedna porucha v jednotce se díky tomu může projevit nejvýše na jednom výstupním signálu EvSSC. Výstup je zakódován kódem 1 z N, porucha se proto může projevit pouze nekódovým slovem. Vstup aktuálního stavu je v kódu 1 z N, každý signál odpovídá právě jednomu stavu. Automaty, pro které lze použít MD architekturu, musejí mít úplně definovanou přechodovou funkci. Nekódové slovo na vstupu aktuálního stavu (způsobené poruchou v ExNSL, poruchou v klopných obvodech nebo poruchou na vstupu aktuálního stavu této jednotky) se proto vždy projeví nekódovým slovem na výstupu jednotky EvSSC. Jednotka EvSSC je tedy bezpečná proti poruchám typu t s výjimkou poruch na vstupech odpovídajících primárním vstupům sekvenčního obvodu. Navíc zajišťuje kontrolu kódu vnitřního stavu. Jestliže jednotka nebude obsahovat redundantní části, bude samočinně testovatelná.

PTC

Vstup i výstup jednotky PTC je zakódován kódem 1 z N. V jednotce opět nejsou sdíleny žádné prostředky, a proto se porucha buď projeví nekódovým slovem a nebo se neprojeví. Implementací je zajištěno, že nekódové slovo na vstupu způsobí nekódové slovo na výstupu jednotky. Jednotka PTC je tedy bezpečná proti poruchám typu t včetně poruch na jejím vstupu. Dále provádí kontrolu kódu na výstupu jednotky EvSSC. Implementací pomocí jednotek O-Cell z obrázku 12 zajišťuje samočinnou testovatelnost jednotky.

ExNSL

Vstup i výstup jednotky ExNSL je také zakódován kódem 1 z N. Jednotka je implementována pomocí logických součtů, logický součet je monotonní funkce, a proto se poruchy v této jednotce (včetně poruch na vstupech jednotky) buď projeví nekódovým slovem na výstupu, nebo se neprojeví Vzhledem k implementaci je jednotka samočinně testovatelná kódem 1 z N.

ExOUT

Jednotka ExOUT nepoužívá na výstupu žádný bezpečnostní kód. Kontrolu správnosti výstupního slova provádí hlídač MD-checker. Implementace této jednotky zajišťuje projevení všech poruch v jednotce na jejím výstupu, nemůže se v ní tedy objevit nedetekovatelná porucha.

EvCh

Použitím dvoudrátové logiky a kódu 1 z N na výstupu jednotky je zajištěna bezpečnost proti poruchám. Vstup jednotky není zakódován, a jestliže vlivem poruchy vzniklo jiné přípustné slovo (tzn. slovo, které patří do množiny výstupních slov obvodu, ale nemělo by se v daném okamžiku objevit na výstupu obvodu), potom bude na výstupu nesprávné kódové slovo. Toto slovo se ale nebude shodovat se slovem z kontrolního výstupu C a chyba bude detekována jednotkou Match detector. Poruchy na vstupech termu jednotky A-Cell nemusí být samočinně testovatelné (Jedná se o poruchy na pinech a, b a c na obrázku 13.)

Match detector

Stejně jako v případě EvCh je i u jednotky Match detector použitím dvoudrátové logiky zajištěna bezpečnosti proti poruchám, a to včetně poruch na vstupech obvodu. Všechny poruchy se projeví nekódovým slovem na výstupu, a proto je tato jednotka úplně samočinně kontrolovaná.

ExCh

Tato jednotka zpracovává odděleně signály S_i a V_i ze všech dvojic (S_i, V_i). Signály S_i představují slovo v kódu 1 z N a signály V_i slovo v kódu (n-1) z n. Implementace pomocí buněk O-Cell zajišťuje úplnou samočinnou kontrolu této jednotky.

Obvod podle MD architektury je tedy bezpečný proti všem poruchám typu t a proti poruchám typu SEU v konfigurační paměti LUT a v paměti aktuálního stavu obvodu. Dále je samočinně testovatelný pro všechny poruchy typu t s výjimkou poruch na primárních vstupech a některých poruch na vstupech buněk A-Cell jednotky EvCh. Obvod nemůže být samočinně testovatelný pro poruchy SEU v konfigurační paměti LUT (viz kapitola 2.4)

3.1.4 Velikost obvodu

Dalším důležitou vlastností je výsledná velikost obvodu. Autoři v [8] uvádějí výsledky pro některé benchmarky. Tyto hodnoty jsou uvedeny v tabulce 6. Význam jednotlivých sloupců tabulky:

- L – počet vstupních signálů obvodu
- N – počet výstupních signálů obvodu
- R – počet různých vnitřních stavů
- H – počet termů (počet přechodů)
- M – počet různých výstupních slov
- S_O – počet CLB potřebných pro implementaci původního, nezabezpečeného obvodu
- S_{MD} – počet CLB potřebných pro implementaci TSC verze sekvenčního obvodu
- Ω_{MD} – nárůst velikosti obvodu oproti nezabezpečené variantě $\Omega_{MD} = (S_{MD} - S_O) / S_O$

Tabulka 6 - výsledná velikost obvodu podle MD architektury

Jméno	*L*	*N*	*R*	*H*	*M*	S_O	S_{MD}	Ω_{MD}
big	18	28	17	185	17	124	251	**100%**
bs	19	13	17	185	17	125	202	**62%**
acdl	16	27	22	214	23	158	317	**101%**
cow	49	24	24	261	18	262	402	**53%**
v1_6	14	18	17	169	17	74	146	**97%**
v1_10	15	18	18	264	18	102	187	**83%**
v11_20	14	29	18	367	17	110	247	**125%**

Průměrný nárůst velikosti obvodu dosahoval 89%.

3.2 Návrh využívající kódy pro detekci jednosměrných chyb

V této kapitole se budeme zabývat návrhem samočinně kontrolovaného obvodu s využitím bezpečnostních kódů schopných detekovat jednosměrné chyby a implementaci obvodu bez použití invertorů (viz [12] a [13]). Při standardní implementaci se jedna porucha může projevit na více výstupních signálech a s různou polaritou (tzn. může změnit původní hodnotu logická 0 na hodnotu logická 1 nebo naopak). Jestliže ale bude implementací zajištěno, aby se každá porucha z dané množiny buď neprojevila nebo projevila jako jednosměrná chyba (tzn. všechny změny na výstupních signálech mají v daném okamžiku stejnou polaritu) na výstupu obvodu, bude každá taková porucha díky vlastnostem použitého bezpečnostního kódu detekovatelná a obvod bude bezpečný proti poruchám.

3.2.1 PLA popis pro implementaci bez invertorů

V [12] a [13] se zabývají takovou úpravou PLA popisu obvodu, aby nebylo nutné použít invertory ani na primárních vstupech. Obvod pak bude bezpečný proti všem poruchám typu t včetně poruch na vstupu obvodu. Při implementaci v FPGA bude bezpečný i proti poruchám způsobujícím chybnou funkci LUT.

Vstup i výstup obvodu je zabezpečen redukovaným kódem M z N (viz kapitola 3.2.2), pro zakódování vnitřního stavu obvodu je použit kód 1 z N. Autoři uvádějí i další způsob rozšíření vstupu umožňující převod do PLA^U. Obě metody způsobí přibližně stejné rozšíření vstupu, použití kódu M z N je výhodnější při propojení více zabezpečených automatů, protože vstup i výstup komunikujících automatů bude ve stejném kódu.

Po rozšíření vstupu lze všechny hodnoty log. 0 ve vstupních slovech a ve slovech vnitřního stavu nahradit nedefinovanou hodnotou, aniž by se tím jakkoliv změnila funkce automatu. Upravený popis je označován jako PLA^U.

V tabulce 7 je příklad automatu podle benchmarku mc.kiss2.

Tabulka 7 - STG popis automatu podle mc.kiss2

X	Q_i	Q_{i+1}	Y
0 - -	1	1	0 0 0 1 0
- 0 -	1	1	0 0 0 1 0
1 1 -	1	2	1 0 0 1 0
- - 0	2	2	0 0 1 1 0
- - 1	2	3	1 0 1 1 0
1 0 -	3	3	0 1 0 0 0
0 - -	3	4	1 1 0 0 0
- 1 -	3	4	1 1 0 0 0
- - 0	4	4	0 1 0 0 1
- - 1	4	1	1 1 0 0 1

V tabulce 8 je popis automatu z tabulky 7 po zakódování vstupu, výstupu a stavové proměnné automatu. Pro zakódování stavu byl použit kód 1 z N, v tomto případě tedy 1 ze 4. Pro zakódování vstupu a výstupu je použit redukovaný kód M z N, konkrétně pro vstup kód 3 z 6 a pro výstup kód 3 ze 7.

Tabulka 8 - PLA popis automatu podle mc.kiss2 se zakódovaným vstupem, výstupem a stavem.

X	Q_i	Q_{i+1}	Y
0 - - 1 - -	1 0 0 0	1 0 0 0	0 0 0 1 0 1 1
- 0 - - 1 -	1 0 0 0	1 0 0 0	0 0 0 1 0 1 1
1 1 - 0 0 -	1 0 0 0	0 1 0 0	1 0 0 1 0 1 0
- - 0 - - 1	0 1 0 0	0 1 0 0	0 0 1 1 0 0 1
- - 1 - - 0	0 1 0 0	0 0 1 0	1 0 1 1 0 0 0
1 0 - 0 1 -	0 0 1 0	0 0 1 0	0 1 0 0 0 1 1
0 - - 1 - -	0 0 1 0	0 0 0 1	1 1 0 0 0 0 1
- 1 - - 0 -	0 0 1 0	0 0 0 1	1 1 0 0 0 0 1
- - 0 - - 1	0 0 0 1	0 0 0 1	0 1 0 0 1 1 0
- - 1 - - 0	0 0 0 1	1 0 0 0	1 1 0 0 1 0 0

V tabulce 9 je výsledný PLA^U popis, který vznikl nahrazením všech hodnot log. 0 ve vstupních a stavových slovech nedefinovanou hodnotou.

Tabulka 9 - PLA^U popis automatu podle mc.kiss2.

X	Q_i	Q_{i+1}	Y
- - - 1 - -	1 - - -	1 0 0 0	0 0 0 1 0 1 1
- - - - 1 -	1 - - -	1 0 0 0	0 0 0 1 0 1 1
1 1 - - - -	1 - - -	0 1 0 0	1 0 0 1 0 1 0
- - - - - 1	- 1 - -	0 1 0 0	0 0 1 1 0 0 1
- - 1 - - -	- 1 - -	0 0 1 0	1 0 1 1 0 0 0
1 - - - 1 -	- - 1 -	0 0 1 0	0 1 0 0 0 1 1
- - - 1 - -	- - 1 -	0 0 0 1	1 1 0 0 0 0 1
- 1 - - - -	- - 1 -	0 0 0 1	1 1 0 0 0 0 1
- - - - - 1	- - - 1	0 0 0 1	0 1 0 0 1 1 0
- - 1 - - -	- - - 1	1 0 0 0	1 1 0 0 1 0 0

V [14] je uveden způsob implementace v FPGA. Funkce popisující automat jsou převedeny na binární rozhodovací diagramy (BDD) a ty jsou poté namapovány na LUT.

Porucha na primárním vstupu ani porucha na stavových signálech se nemůže projevit nesprávným kódovým slovem na výstupu. Obvod podle PLA^U je tak odolný proti poruchám typu t včetně poruch na primárním vstupu obvodu. Při implementaci v FPGA je také odolný proti poruchám SEU v konfigurační paměti LUT a v klopných obvodech pro uložení aktuálního stavu obvodu.

Při implementaci v FPGA bude opět problém s dosažením samočinné kontroly obvodu. Pro otestování všech poruch SEU v LUT by bylo nutné provést pro každou LUT triviální test.

3.2.2 Redukovaný kód M z N

Výhodou kódů M z N je schopnost detekovat jednosměrné chyby. Tento kód je ale nesystematický, a proto jej nelze použít pro kódování výstupů obvodu. V [15] byl popsán systematický kód M z N, nazývaný redukovaný kód M z N (reduced m out of n code).

Proces kódování je náročnější než u jiných systematických kódů. Uvažujme množinu O všech přípustných slov. Kódování vychází z následující definice:

Dvě proměnné (bity) Z_i a Z_j jsou neslučitelné, jestliže mají obě současně v alespoň jednom kódovém slově o, $o \in O$, hodnotu logická 1. V opačném případě jsou slučitelné.

Množina slučitelných (neslučitelných) proměnných je množina proměnných, ve které jsou všechny dvojice Z_i a Z_j, $i \neq j$, slučitelné (neslučitelné).

Z těchto definic vyplývá, že ze všech proměnných patřících do jedné množiny slučitelných proměnných má v každém výstupním slově nejvýše jedna hodnotu logická 1.

Množina proměnných M je rozdělena do několika podmnožin slučitelných proměnných T_i (tyto podmnožiny by měly být co největší), jejichž průnik musí být prázdný a sjednocení odpovídá původní množině proměnných M. Každé podmnožině T_i je přiřazena jedna kontrolní proměnná c_i, která nabývá hodnoty logická 1 v případě, kdy tuto hodnotu nemá žádná jiná proměnná z podmnožiny T_i.

Přiřazením kontrolního bitu byla každá množina doplněna na kód 1 z N. Počet podmnožin T_i (a tím i počet kontrolních bitů) závisí na množině původních slov. V nejlepším případě budou všechny proměnné slučitelné, bude potřeba nejvýše jeden kontrolní bit a výsledný kód bude shodný s kódem 1 z N. V nejhorším případě bude každá podmnožina T_i obsahovat právě jednu proměnnou, kontrolních proměnných bude stejný počet jako informačních proměnných a výsledný kód bude odpovídat kódu N z 2N.

Příklad redukovaného kódu M z N je v tabulce 10. První podmnožina T_1 obsahuje proměnné Z_1, Z_3 a Z_4 a je jí přiřazen kontrolní bit c_1, druhá podmnožina T_2 obsahuje proměnnou Z_2 a náleží jí kontrolní bit c_2 a třetí podmnožina T_3 obsahuje proměnnou Z_5 a náleží jí poslední kontrolní bit c_3.

Tabulka 10 - Redukovaný kód M z N

	Informační proměnné					***kontrolní proměnné***		
	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
	Z_1	Z_2	Z_3	Z_4	Z_5	c_1	c_2	c_3
o_1	0	0	0	0	0	1	1	1
o_2	1	1	0	0	1	0	0	0
o_3	0	1	0	1	0	0	0	1
o_4	0	0	1	0	1	0	1	0
o_5	0	0	1	0	0	0	1	1
o_6	1	1	0	0	0	0	0	1
o_7	0	1	0	0	0	1	0	1

V experimentu v kapitole 5.1 jsme provedli porovnání redukovaného kódu M z N s Bergerovým kódem z hlediska počtu bitů potřebných pro zabezpečení výstupu obvodu.

3.2.3 Hlídač redukovaného kódu M z N

V [15] je uveden způsob implementace hlídače kódu v FPGA. Při kontrole kódu se využívá toho, že redukovaný kód M z N se skládá z několika skupin tvořících kód 1 z N. Tyto skupiny jsou hlídány samostatně a dílčí výsledky jsou zpracovány standardním způsobem do jediné dvojice indikující stav obvodu. Kontrola kódu je díky tomu jednodušší než kontrola Bergerova kódu, kde je nutné zjistit počet bitů s hodnotou log 0 a porovnat s kontrolní částí kódu.

Hlídač jedné skupiny (hlídač kódu 1 z N) je realizován pomocí prahové funkce F_k, kde k vyjadřuje podmínku, při jejímž splnění má prahová funkce hodnotu logická 1. Výstupy hlídače jedné skupiny

lze vyjádřit následovně:

$R_1 = F_{>=1}(X_1) + F_{>=2}(X_2)$

$R_2 = F_{>=2}(X_1) + F_{>=1}(X_2)$

kde X_1 a X_2 jsou dvě disjunktní podmnožiny množiny X, $X = X_1 \cup X_2$.

V případě, že počet prvků množiny X je menší než počet vstupů LUT, potřebujeme pro realizaci těchto funkcí dvě LUT. V opačném případě použijeme dekompozici (a počet LUT bude samozřejmě větší):

$F_{=1}(X) = F_{=1}(F_{=1}(X_1), F_{=1}(X_2))$

kde stejně jako v předchozím případě X_1 a X_2 jsou disjunktní podmnožiny množiny X, $X = X_1 \cup X_2$.

Pro každou skupinu proměnných B_i tak získáme dva signály, $R_{1\,i}$ a $R_{2\,i}$. Výsledné funkce R_1 a R_2 získáme kaskádním zapojením funkce definované takto:

$R_1 = R_{1i} R_{1j} + R_{2i} R_{2j}$

$R_2 = R_{1i} R_{2j} + R_{2i} R_{1j}$

Takto implementovaný hlídač bude pro poruchy typu t úplně samočinně kontrolovaný. Bude také bezpečný proti všem poruchám SEU v konfigurační paměti LUT.

3.3 Porovnání MD architektury a návrhu s kódy pro detekci jednosměrných chyb

Oba způsoby budou při implementaci v FPGA bezpečné proti poruchám typu SEU v konfigurační paměti LUT a v klopných obvodech , které jsou součástí navrhovaného obvodu a dále proti poruchám typu t na vstupních a výstupních pinech CLB.

MD architektura je vhodná pro obvody, které mají malou množinu výstupních slov. Při testech (viz [8]) na několika standardních benchmarcích dosahoval průměrný nárůst velikosti obvodu hodnoty 89%. Nevýhodou MD architektury je nestandardní implementace (struktura obvodu je přesně definována).

Návrh využívající kódy pro detekci jednosměrných chyb používá místo strukturního popisu funkční popis. Obvod je bezpečný proti poruchám nezávisle na implementaci. Díky tomu je návrh touto metodou jednodušší.

Obvod podle MD architektury je bezpečný proti poruchám a také samočinně testovaný pro většinu poruch typu t, ale není úplně samočinně kontrolovaný. Vlastnosti obvodu podle návrhu využívajícího kódy pro detekci jednosměrných chyb ověříme experimentem (viz kapitola 5.3).

4 Vlastní řešení

V této kapitole jsou popsány algoritmy a postupy, které jsme použili v nástrojích pro ověření TSC vlastností obvodu. Všechny nástroje byly naprogramovány v jazyce Java s využitím vývojového prostředí Borland JBuilder 9.0 Personal. V první části jsou uvedeny formáty použitých souborů a způsob jejich překladu do vnitřní formy. Tři uvedené formáty jsou standardní, dva vznikly pro účely této práce. Dále je uveden způsob kódování redukovaným kódem M z N. Redukovaný kód M z N je systematický, je schopný detekovat jednosměrné chyby a způsob jeho kontroly je jednodušší než způsob kontroly Bergerova kódu. Proto se jím budeme zabývat podrobněji. Porovnání s Bergerovým kódem je provedeno v experimentu v kapitole 5.1. Dále je v této kapitole uveden překlad z formátu KISS2, který slouží jako vstupní formát, do formátu PLA, který je používán nástroji pro minimalizaci, a překlad z formátu PLA do formátu VHDL, který lze zpracovat nástrojem pro syntézu obvodů do FPGA. Takto jsme schopni převést soubor z původního zadání až do popisu mapování na jednotlivé LUT v FPGA. V kapitole 4.5 je popsán způsob generování testů pro sekvenční obvody. Na rozdíl od testů pro kombinační obvody obsahují tyto testy posloupnost testovacích a nastavovacích vektorů, která zajistí otestování všech přechodů v automatu. V další kapitole je popsán vlastní simulátor poruch – FE-SIM. Důvodů pro návrh vlastního simulátoru bylo několik. Potřebovali jsme simulátor poruch pro sekvenční obvody; popis testovaného obvodu měl být ve formátu EDIF (ten přímo odpovídá namapování funkcí na LUT v FPGA) a chyby způsobené poruchami měly být zpracovány z hlediska samočinné kontroly (tzn včetně kontroly kódu) a ne pouze z hlediska testovatelnosti poruch. Těmto požadavkům neodpovídal žádný dostupný simulátor poruch (např. FSIM nebo HOPE (oba viz [16]). Simulátor poruch FE-SIM umožňuje simulovat poruchy v sekvenčních i kombinačních obvodech, vstupním formátem pro popis obvodu je formát EDIF a chyby vyhodnocuje z hlediska samočinné kontroly obvodu. V poslední podkapitole je popsáno rozdělení poruch z hlediska on-line testování a detekce.

Zdrojové texty pro uvedené algoritmy a postupy, hotové nástroje a uživatelské příručky jsou na přiloženém CD-ROM.

4.1 Formáty vstupních a výstupních dat

V této podkapitole je uveden popis použitých formátu. Pro formáty KISS2, PLA, EDIF a TST jsou popsány způsoby překladu do vnitřní formy. Formát KISS2 je formát pro popis automatů v benchmarcích. Dále je popsán formát PLA, který je používán nástroji pro minimalizaci a formát EDIF, což je jeden z výstupních formátů nástroje Xilinx ISE pro syntézu obvodů v FPGA. Jako poslední jsou uvedeny formát pro popis testu (TST) a formát pro popis výsledků testu (RES). Oba tyto formáty byly navržené v pro účely této práce.

4.1.1 Formát pro popis automatu - KISS2

Jako formát vstupních dat pro naše experimenty používáme formát KISS2. Popis tohoto formátu je uveden ve specifikaci formátu BLIF [17]. V námi používaných souborech chyběl startovní (.start_kiss) a ukončovací (.end_kiss) symbol, proto jsme je označili jako nepovinné.

Formát KISS2 vypadá takto:

```
[.start_kiss]
.i <num-inputs>
.o <num-outputs>
[.p <num-terms>]
[.s <num-states>]
[.r <reset-state>]
<input> <current-state> <next-state> <output>
...
<input> <current-state> <next-state> <output>
[.end_kiss]
```

kde *num-inputs* je počet vstupů automatu, *num-outputs* je počet výstupů automatu, *num-terms* je počet přechodů v automatu (počat řádků přechodové tabulky), *num_states* je počet různých stavů automatu a *reset-state* je počáteční stav obvodu. V popisu přechodu představuje *input* vstupní vektor, *current-state* jméno aktuálního stavu, *next-state* jméno následujícího stavu a *output* výstupní vektor. Vstupní a výstupní vektory jsou posloupnost hodnot z množiny {0,1,-}.

Gramatiku formátu KISS2 lze definovat takto:

- množina neterminálních symbolů N:
  {S, IN, OUT, PSR, SR, R, STATE1, STATE, STATE_END}
- množina terminálních symbolů T.
  {lex_kw_start_kiss, lex_kw_i, lex_kw_o, lex_kw_p, lex_kw_s, lex_kw_r, lex_kw_end_kiss, lex_str, lex_asterisk}
- množina pravidel P
  S → lex_kw_start_kiss IN
  S → IN
  IN → lex_kw_i lex_str OUT
  OUT → lex_kw_o lex_str PSR
  PSR → lex_kw_p lex_str SR
  PSR → SR
  SR → lex_kw_s lex_str R
  SR → R
  R → lex_kw_r lex_str STATE1
  R → STATE1
  STATE1 → lex_str STATE_END

STATE → lex_str STATE_END

STATE → lex_kw_end_kiss

STATE_END → lex_str lex_str lex_str STATE

STATE_END → lex_asterisk lex_str lex_str

STATESTATE → ε

- počáteční symbol:

  S

Gramatiku budeme využívat při překladu do vnitřní formy. Vnitřní forma je v tomto případě seznam čtveřic – (vstupní vektor, aktuální stav, následující stav, výstupní vektor). Všechna uživatelská data, jako jsou čísla, binární vektory a textové řetězce zastupuje terminální symbol lex_str. Atributy tohoto symbolu jsou uvedeny v tabulce 11.

Tabulka 11 - atributy terminálu lex_str pro formát KISS2

atribut	***typ atributu***	***význam atributu***
value	String	původní hodnota terminálu
isBin	boolean	příznak, zda terminál představuje binární vektor
isNumber	boolean	příznak, zda terminál představuje číslo

Množina sémantických pravidel pro překlad do vnitřní formy je v tabulce 12. Používáme v nich dvě globální proměnné: G_INPUT_N obsahující počet bitů vstupního vektoru a G_OUTPUT_N obsahující počet bitů výstupního vektoru. Globální funkce SEMANT_ERROR() představuje signalizaci chyby při překladu. Objekt označený jako MACHINE obsahuje vytvářenou vnitřní reprezentaci a v sémantických pravidlech se používají jeho metody pro nastavení počtu vstupů (setInputN()), nastavení počtu výstupů (setOutputN()), nastavení startovního stavu (setStartState()) a metoda pro přidání přechodu do tabulky přechodů (addTransition()). Sémantická pravidla jsou napsána v jazyce Java.

Tabulka 12 - sémantická pravidla pro překlad do vnitřní formy pro formát KISS2

pravidlo gramatiky	***sémantická pravidla***
S → lex_kw_start_kiss IN	-
S → IN	-
IN → lex_kw_i lex_str OUT	if (!lex_str.isNumber) SEMANT_ERROR(); G_INPUT_N = (int) lex_str.value; MACHINE.setInputN(G_INPUT_N);
OUT → lex_kw_o lex_str PSR	if (!lex_str.isNumber) SEMANT_ERROR(); G_OUTPUT_N = (int) lex_str.value; MACHINE.setOutputN(G_OUTPUT_N);
PSR → lex_kw_p lex_str SR	if (!lex_str.isNumber) SEMANT_ERROR();

pravidlo gramatiky	*sémantická pravidla*
PSR → SR	-
SR → lex_kw_s lex_str R	if (!lex_str.isNumber) SEMANT_ERROR();
SR → R	-
R → lex_kw_r lex_str STATE1	if (!lex_str.isNumber) SEMANT_ERROR(); MACHINE.setStartState(lex_str.value)
R → STATE1	-
STATE1 → lex_str STATE_END	if (!lex_str.isBin && lex_str.value.length != G_INPUT_N) SEMANT_ERROR(); INPUT = lex_str.value;
STATE → lex_str STATE_END	if (!lex_str.isBin && lex_str.value.length != G_INPUT_N) SEMANT_ERROR(); INPUT = lex_str.value;
STATE_END → lex_str_1 lex_str_2 lex_str_3 STATE	if (!lex_str_3.isBin && lex_str_3.value.length != G_OUTPUT_N) SEMANT_ERROR(); MACHINE.addTransition(INPUT, lex_str_1.value, lex_str_2.value, lex_str_3.value);
STATE_END → lex_asterisk lex_str_1 lex_str_2	if (!lex_str_2.isBin && lex_str_2.value.length != G_OUTPUT_N) SEMANT_ERROR(); MACHINE.addTransition(INPUT, null, lex_str_2.value, lex_str_3.value);
STATE → lex_kw_end_kiss	-
STATE →	-

4.1.2 Formát pro popis kombinačních obvodů – PLA

Formát PLA se používá k popisu kombinačních obvodů. Podrobný popis formátu je uveden například v [18]. Popis podle formátu PLA může obsahovat řadu parametrů, pro naše účely postačuje jen malá podmnožina těchto parametrů. Tomu byl přizpůsoben i překladač do vnitřní formy.

Funkce jsou ve formátu PLA definovány pomocí pravdivostní tabulky. Tabulku lze rozdělit do dvou polí. V prvním (levém) poli odpovídá každý sloupec jednomu bitu vstupu, v druhém (pravém) poli odpovídá každý sloupec jednomu bitu výstupu. Levá část každého řádku tabulky představuje jeden minterm (součin libovolných vstupních bitů buď v přímé, nebo invertované podobě). Význam pravé části závisí na typu popisu. V případě popisu typu *fd* hodnota 1 ve sloupci *i* a řádku *j* znamená, že jestliže bude mít minterm *j* hodnotu log. 1, bude mít i výstupní bit *i* hodnotu log. 1. Hodnotal 0 a - (nedefinováno) nemá žádný význam. V druhém případě, při popisu typu *fr*, hodnota 1 (0) ve sloupci *i* a řádku *j* znamená, že jestliže bude mít minterm *j* hodnotu log. 1 (log. 0), bude mít i výstupní bit *i* hodnotu log. 1 (log. 0). Hodnota - (nedefinováno) nemá žádný význam.

Formát PLA používaný v této práci má následující strukturu:

```
.i <num-inputs>
.o <num-outputs>
[.p <num-terms>]
[.ilb <input-names>]
[.ob <output-names>]
[.type <type>]
<input> <output>
...
<input> <output>
[.e]
```

kde *num-inputs* je počet bitů vstupního vektoru, *num-outputs* je počet bitů výstupního vektoru, *num-terms* je počet termů (počet řádků pravdivostní tabulky), *input-names* je pole jmen vstupních bitů, *output-names* je pole jmen výstupních bitů a *type* je typ PLA. *Input* je vstupní vektor a *output* je výstupní vektor.

Při popisu parametrů nebudeme požadovat přesné pořadí parametrů, což povede ke snížení složitosti gramatiky za cenu přidání několika sémantických pravidel. Gramatiku formátu PLA lze pak definovat takto:

- množina neterminálních symbolů N:
  {S, NEXPR}
- množina terminálních symbolů T.
  {lex_kw_i, lex_kw_o, lex_kw_p, lex_kw_ilb, lex_kw_ob, lex_kw_type, lex_kw_e, lex_str}
- množina pravidel P
  S → lex_kw_o lex_str S
  S → lex_kw_p lex_str S
  S → lex_kw_ilb lex_str S
  S → lex_kw_ob lex_str S
  S → lex_kw_type lex_str S
  S → lex_str lex_str NEXPR
  NEXPR → lex_str lex_str NEXPR
  NEXPR → lex_kw_e
  NEXPR → ε
- počáteční symbol:
  S

Vnitřní forma je seznam dvojic (input-vector, output-vector). Překlad do vnitřní formy je podobný překladu použitého u formátu KISS2.

4.1.3 Formát pro popis obvodů – EDIF

Formát EDIF (Electronic Design Interchange Format) je formát používaný k přenosu dat mezi různými CAD systémy pro návrh obvodů. Popis formátu (gramatika) je uvedena například v[19]. V této práci budeme používat EDIF verze 2 0 0.

Jedná se o výrazně komplikovanější formát než ostatní uváděné formáty. Pro překlad do vnitřní formy jsme použili parser z [20]. V následujících odstavcích popíšeme tuto metodu.

Překlad do námi požadované vnitřní formy se skládá ze dvou částí. Prvním krokem je překlad do vnitřní formy, která představuje derivační strom načteného řetězce. Druhým krokem je procházení tohoto derivačního stromu do hloubky a vytváření vlastní vnitřní formy, odpovídající konkrétním požadavkům (v našem případě popisu obvodu podle kapitoly 4.6). Díky tomu je možno definovat pouze třídu pro průchod stromem (tato třída je označována jako Visitor) a není nutné upravovat třídy odpovídající jednotlivým terminálním a neterminálním symbolům (objekty těchto tříd jsou uzly derivačního stromu).

Všechny třídy odpovídající terminálním a neterminálním symbolům mají společného předka (nazveme jej *NodeToken*). Aby bylo možné při průchodu stromem určit typ aktuálně zpracovávaného uzlu, má každá třída dvojici metod *accept()* (Druhá metoda *accept()* je určena pro atributový překlad.):

```
public void accept(visitor.Visitor v) {
  v.visit(this);
}
public Object accept(visitor.ObjectVisitor v, Object argu) {
  return v.visit(this,argu);
}
```

Třída *Visitor* obsahuje metodu *visit()* přetíženou pro všechny potomky třídy *NodeToken*. Zavoláním konkrétní metody z uzlu stromu je určen typ uzlu. Metoda *visit()* obsahuje další volání metod *accept()* pro následníky uzlu daného typu a metody pro vytváření vlastní vnitřní formy. Tak je zajištěn průchod stromem do hloubky.

Fragment kódu využívajícího tuto metodu může vypadat například takto:

```
...
EdifVisitor myVisitor = new EdifVisitor();
EdifParserCore parser = new EdifParserCore(new FileInputStream(filename));
Node root = parser.edif();
root.accept(myVisitor, argument);
...
```

Nejprve je vytvořen objekt pro průchod stromem do hloubky (*myVisitor*). Objekt *parser* představuje překladač do vnitřní formy – derivačního stromu. Překlad se provede zavoláním metody *edif()*, která vrací kořen derivačního stromu (objekt *root*). Zavoláním metody accept() na kořen stromu (*root*) je proveden průchod stromem do hloubky a vytvoření vlastní vnitřní reprezentace.

4.1.4 Formát pro popis testu – TST

Součástí návrhu simulátoru FE-SIM (viz kapitola 4.6) byl i návrh formátu pro popis testu. Simulátor umožňuje testovat jak sekvenční, tak kombinační obvody. Kromě kontroly, zda vypočítaný vektor odpovídá požadovanému vektoru také detekuje, jestli vektor je nebo není kódovým slovem použitého bezpečnostního kódu. Popis testu by tak měl obsahovat:

- typ testu (test pro sekvenční nebo kombinační obvod)
- délku vstupních, výstupních a případně stavových vektorů
- použité kódování vektorů (kód a bity, které tvoří kódové slovo). Lze použít několik různých kódů pro libovolné skupiny bitů vektoru, libovolný bit vektoru může být součástí více kódových skupin nebo nemusí patřit do žádné.
- posloupnost kroků testu. Test pro kombinační obvody se skládá pouze z testovacích kroků, každý krok testu obsahuje vstupní vektor a odpovídající výstupní vektor. Pro sekvenční obvody se test skládá z testovacích a nastavovacích kroků. Testovací krok obsahuje vstupní vektor a odpovídající výstupní a stavový vektor. Nastavovací kroky slouží pro nastavení obvodu do požadovaného stavu, obsahují proto pouze vstupní vektory. Mezi nastavovací kroky testu lze považovat i krok s příkazem pro reset obvodu (nastavení vnitřního stavu obvodu do startovního stavu).

Snažili jsme se o dosažení jednodušší gramatiky i za cenu složitějších sémantických pravidel. Gramatika pro formát TST je definována následovně:

- množina neterminálních symbolů N:
  {S0, S1, S, CODE_TYPE, VECTOR_TEST_M, VECTOR_TEST_C, VECTOR_SET, VECTOR_RESET, NEXT_VECTOR}
- množina terminálních symbolů T.
  {lex_kw_i, lex_kw_o, lex_kw_s, lex_kw_t, lex_kw_n, lex_kw_type, lex_kw_c, lex_kw_r, lex_kw_inum, lex_kw_onum, lex_kw_snum, lex_str}
- množina pravidel P
  S0 → lex_kw_type lex_str S1
  S1 → lex_kw_inum lex_str S1
  S1 → lex_kw_onum lex_str S1
  S1 → lex_kw_snum lex_str S1
  S1 → S
  S → lex_kw_i CODE_TYPE S
  S → lex_kw_o CODE_TYPE S
  S → lex_kw_s CODE_TYPE S

CODE_TYPE → lex_str lex_str lex_str

S → NEXT_VECTOR

VECTOR_TEST_M → lex_kw_t lex_str lex_str lex_str NEXT_VECTOR

VECTOR_SET → lex_kw_n lex_str NEXT_VECTOR

VECTOR_TEST_C → lex_kw_c lex_str lex_str NEXT_VECTOR

VECTOR_RESET → lex_kw_r NEXT_VECTOR

NEXT_VECTOR → VECTOR_TEST_T

NEXT_VECTOR → VECTOR_SET

NEXT_VECTOR → VECTOR_TEST_C

NEXT_VECTOR → VECTOR_RESET

NEXT_VECTOR → ε

- počáteční symbol:

  S0

Gramatiku budeme využívat při překladu do vnitřní formy. Všechna uživatelská data, jako jsou čísla, binární vektory, textové řetězce a pole čísel zastupuje terminální symbol lex_str. Atributy tohoto symbolu jsou uvedeny v tabulce 13. Neterminální symbol CODE_TYPE obsahuje dědičný atribut *vectorLength*, který obsahuje délku kódového slova a syntetizovaný atribut *description*, který bude obsahovat definici kontrolního kódu (typ kódu a bity, které tvoří kódové slovo).

Tabulka 13 - atributy terminálu lex_str pro formát TST

atribut	***typ atributu***	***význam atributu***
value	String	původní hodnota
isBin	boolean	příznak, zda terminál představuje binární vektor
isNumber	boolean	příznak, zda terminál představuje číslo
isByteArray	boolean	příznak, zda terminál představuje pole čísel (posloupnost čísel oddělených čárkou)
number	int	řetězec value převedený na číslo (tento atribut je platný pouze tehdy, když je nastaven příznak isNumber)
byteValues	byte[]	pole celočíselných hodnot (tento atribut je platný pouze tehdy, když je nastaven příznak isByteArray)

Množina sémantických pravidel pro překlad do vnitřní formy je v tabulce 14. Jsou použity globální proměnné pro uložení délky vstupního (*G_INPUT_N*), výstupního (*G_OUTPUT_N*) a stavového vektoru (*G_STATE_BITS_N*) . Další pomocnou proměnnou je proměnná pro uložení typu testu (*G_TEST_TYPE*). Jsou podporovány dva typy testu, test pro kombinační obvody, označovaný jako TEST_TYPE_COMB a test pro sekvenční obvody TEST_TYPE_MACHINE.

Tabulka 14 - zjednodušená sémantická pravidla pro překlad formátu TST do vnitřní formy

pravidlo gramatiky	***sémantická pravidla***
S0 → lex_kw_type lex_str S1	G_TEST_TYPE = (int) lex_str.value; if (G_TEST_TYPE != TEST_TYPE_COMB && G_TEST_TYPE != TEST_TYPE_MACHINE) SEMANT_ERROR(); TESTGEN.setType(G_TEST_TYPE);
S1 → lex_kw_inum lex_str S1	if (!lex_str.isNumber) SEMANT_ERROR(); G_INPUT_N = lex_str.number; TESTGEN.setInputN(G_INPUT_N);
S1 → lex_kw_onum lex_str S1	if (!lex_str.isNumber) SEMANT_ERROR(); G_OUTPUT_N = lex_str.number; TESTGEN.setOutputN(G_OUTPUT_N);

S1 → lex_kw_snum lex_str S1	if (!lex_str.isNumber) SEMANT_ERROR();
	if (G_TEST_TYPE !=TEST_TYPE_MACHINE)
	SEMANT_ERROR();
	G_STATEBITS_N = lex_str.number;
	TESTGEN.setStateBitsN(G_STATEBITS_N);

S1 → S	if (G_INPUT_N < 0 || G_OUTPUT_N < 0 || (G_TEST_TYPE == TEST_TYPE_MACHINE && G_STATEBITS_N < 0)) SEMANT_ERROR();
S → lex_kw_i CODE_TYPE S	CODE_TYPE.vectorLength = G_INPUT_N; TESTGEN.addInputCode(CODE_TYPE.description)
S → lex_kw_o CODE_TYPE S	CODE_TYPE.vectorLength = G_OUTPUT_N; TESTGEN.addOutputCode(CODE_TYPE.description)
S → lex_kw_s CODE_TYPE S	if (G_TEST_TYPE !=TEST_TYPE_MACHINE) SEMANT_ERROR(); CODE_TYPE.vectorLength = G_STATEBITS_N; TESTGEN.addStateCode(CODE_TYPE.description)

CODE_TYPE → lex_str_1 lex_str_2 lex_str_3	if (UNKNOWN_CODE(lex_str_1)) SEMANT_ERROR();
	if (!lex_str_2.isByteArray) SEMANT_ERROR()
	if (!lex_str_3.isByteArray && lex_str_3.value != „NONE“)
	SEMANT_ERROR()
	CODE_TYPE.description = CODE_DEF(lex_str_1.value,
	lex_str_2.byteValues, lex_str_3.byteValues)

VECTOR_TEST_M → lex_kw_t lex_str_1 lex_str_2 lex_str_3 NEXT_VECTOR	if (G_TEST_TYPE !=TEST_TYPE_MACHINE)
	SEMANT_ERROR();
	if (!lex_str_1.isBin || lex_str_1.value.length != G_INPUT_N ||
	!lex_str_2.isBin || lex_str_2.value.length != G_STATEBITS_N ||
	!lex_str_3.isBin || lex_str_3.value.length != G_OUTPUT_N)
	SEMANT_ERROR()
	TESTGEN.addVector_M(lex_str_1.value, lex_str_2.value,
	lex_str_3.value)

VECTOR_SET → lex_kw_n lex_str NEXT_VECTOR	if (G_TEST_TYPE !=TEST_TYPE_MACHINE)
	SEMANT_ERROR();
	if (!lex_str.isBin || lex_str.value.length != G_INPUT_N)
	SEMANT_ERROR()
	TESTGEN.addVector_S(lex_str.value)

VECTOR_TEST_C → lex_kw_c lex_str_1 lex_str_2 NEXT_VECTOR	if (G_TEST_TYPE !=TEST_TYPE_COMB) SEMANT_ERROR
	();
	if (!lex_str_1.isBin || lex_str_1.value.length != G_INPUT_N ||
	!lex_str_2.isBin || lex_str_2.value.length != G_OUTPUT_N)
	SEMANT_ERROR()
	TESTGEN.addVector_M(lex_str_1.value, lex_str_2.value,
	lex_str_3.value)

VECTOR_RESET → lex_kw_r NEXT_VECTOR	if (G_TEST_TYPE !=TEST_TYPE_MACHINE) SEMANT_ERROR(); TESTGEN.addVector_R();

4.1.5 Formát pro popis výsledků simulace – RES

Formát RES byl navržen pro popis výsledků simulace. Pro každou poruchu v obvodu obsahuje počet a případně výpis vektorů rozdělených podle toho, jaké projevení poruchy způsobí. Rozdělení je podrobně popsáno v kapitole 4.7.

Záznam pro jednu poruchu pro sekvenční obvod vypadá následovně:

```
<typ-poruchy> <jméno pinu>@<jméno součástky>
  testVectorsN: <počet testovacích vektorů 1>
  [testVectorsIndexes: <výpis testovacích vektorů 1>]
  test"STATE"VectorsN: <počet testovacích vektorů 2>
  [test"STATE"VectorsIndexes: <výpis testovacích vektorů 2>]
  errorVectorsN: <počet chybových vektorů>
  [errorVectorsIndexes: <výpis chybových vektorů>]
```

Za podrobným výpisem všech poruch následuje celková statistika. Rozdělení poruch do jednotlivých skupin je popsáno v tabulce 17 v kapitole 4.7.

```
Number of faults: <celkový počet poruch v obvodu>
Number of hidden faults (no error): <počet poruch typu A>
Number of detectable faults (not a code word): <počet poruch typu B>
Number of undetectable faults (incorrect codeword): <počet poruch typu C>
Number of detectable and undetectable faults: <počet poruch typu D>
Probability of correct detecting of error: <X>/<Y> (<Z>%)
```

Poslední údaj vyjadřuje poměr případů, kdy se libovolná porucha pro libovolný vektor projeví detekovatelnou chybou (hodnota *X*) ku počtu případů, kdy se porucha projeví (tedy když se projeví detekovatelnou nebo nedetekovatelnou chybou – hodnota *Y*). Procentuální vyjádření obsahuje hodnota *Z*.

4.2 Algoritmus kódování redukovaným kódem M z N

Redukovaný kód M z N byl popsán v kapitole 3.2.2. Bity vektoru jsou rozděleny do skupin a každá skupina tvoří buď samostatně nebo společně s jedním přidaným kontrolním bitem kód 1 z N. Při zabezpečování výstupu obvodu redukovaným kódem M z N je nutné znát celou množinu výstupních slov, pokud se tato množina z jakéhokoliv důvodu změní, je ve většině případů nutné znovu vygenerovat nové zakódování všech slov.

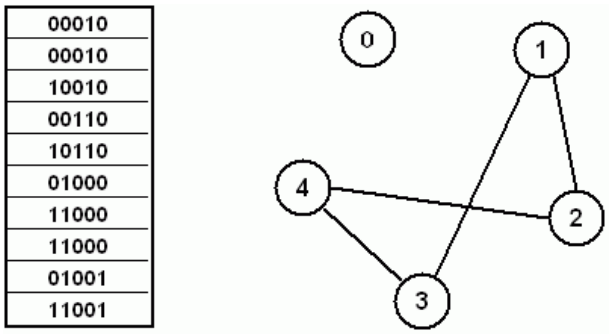
Proces kódování spočívá v rozdělení bitů vektoru do co nejmenšího počtu disjunktních skupin, které tvoří buď samostatně, nebo po doplnění kontrolního bitu kód 1 z N. Postup se skládá z následujících kroků:

- vytvoření grafu slučitelnosti pro bity vektoru. Dva bity jsou slučitelné tehdy, jestliže v každém slově z množiny výstupních slov má nejvýše jeden z těchto bitů hodnotu logická 1. V opačném případě jsou neslučitelné. Uzly grafu odpovídají jednotlivým bitům vektoru. Mezi uzly je hrana tehdy, jsou-li příslušné bity slučitelné.

- Nalezení optimálního rozdělení grafu slučitelnosti na kliky (Clique partitioning). Klika v grafu představuje skupinu bitů, které jsou vzájemně slučitelné. To tedy znamená, že v každém slově má nejvýše jeden bit skupiny hodnotu log. 1. Nalezení optimálního rozdělení je NP-těžký problém. Pro jeho vyřešení je použita Tsengova heuristika (viz [21]).

- Doplnění nedefinovaných bitů. To je provedeno takovým způsobem, aby v každém výstupním slově obsahovala každá skupina slučitelných bitů pouze jediný bit s hodnotou log. 1. Jestliže existuje slovo, ve kterém mají všechny bity některé skupiny hodnotu log. 0, je k dané skupině přidán kontrolní bit. Jeho hodnota je taková, aby společně se skupinou bitů, ke které byl přiřazen, tvořili kód 1 z N.

Při vytváření grafu kompatibility je nejprve vytvořen úplný graf. Jeden uzel grafu odpovídá jednomu bitu vektoru. Pro každé výstupní slovo lze vytvořit množinu B obsahující bity s hodnotou logická 1. Bity množiny B jsou neslučitelné, a proto jsou z grafu odstraněny všechny hrany mezi uzly odpovídajícím bitům množiny B. Po zpracování všech přípustných slov představuje výsledný graf slučitelnosti. Na obrázku 14 je jako příklad uvedena množina výstupních slov a graf slučitelnosti pro benchmark mc.KISS2.



Obrázek 14 - množina výstupních slov a graf slučitelnosti pro benchmark mc.kiss2

Pro nalezení pokrytí grafu klikami je použita Tsengova heuristika. Ta obsahuje následující kroky:

1. Vybrání hrany e = (v, w) s maximální hodnotou $\Pi(e)$,

   $\Pi(e)=|\{u\in V:(u,v)\in E\wedge(u,w)\notin E\}|$.

   Hodnota $\Pi(e)$ představuje počet společných sousedů koncových uzlů v a w hrany e

Je-li takových hran více, vybere se z nich hrana s minimální hodnotou $\mu(e)$,

$\mu(e)=|\{(u,v)\in E:(u,w)\notin E\}\cup\{(u,w)\in E\}|$.

Hodnota $\mu(e)$ představuje počet hran, které budou odstraněny případným sloučením koncových uzlů hrany *e*.

Jestliže více hran splňuje i druhé kritérium, je náhodně vybrána jedna z nich.

2. Sloučení koncových uzlů vybrané hrany do jednoho uzlu. Nový uzel je propojen hranami pouze s těmi uzly, se kterými byly propojeny oba původní uzly.

3. Opakování kroku 1 a 2, dokud graf neobsahuje pouze izolované uzly. Každý izolovaný uzel představuje jednu kliku grafu.

Tsengovou heuristikou byly bity rozděleny do skupin. Aby každá skupina představovala kód 1 z N, je nutné vhodně dodefinovat nedefinované bity (pokud existují) a případně přidat kontrolní bit.

Jako příklad uvedeme zakódování výstupu obvodu podle mc.KISS2. První skupinu T_0 tvoří bit b_0 a jí mu přiřazen kontrolní bit c_0. Druhou skupinu T_1 tvoří bity b_1 a b_3. Bity skupiny T1 představují kód 1 z N, a tak není nutné přidávat kontrolní bit. Poslední skupinu T_2 tvoří bity b_2 a b_4. Této skupině je přiřazen kontrolní bit c_1. Výsledný kód je v tabulce 15.

Tabulka 15 - zakódování výstupu obvodu mc.kiss2 redukovaným kódem M z N

b_0	b_1	b_2	b_3	b_4	c_0	c_1
0	0	0	1	0	1	1
0	0	0	1	0	1	1
1	0	0	1	0	0	1
0	0	1	1	0	1	0
1	0	1	1	0	0	0
0	1	0	0	0	1	1
1	1	0	0	0	0	1
1	1	0	0	0	0	1
0	1	0	0	1	1	0
1	1	0	0	1	0	0

4.3 Převod KISS2 do PLA

Formát KISS2 je velmi podobný formátu PLA. Používá se pro popis sekvenčních obvodů, způsob popisu představuje tabulku přechodů. V tabulce přechodů představují vstupní vektor a aktuální stav vstupy kombinačního obvodu; následující stav a výstupní vektor výstupy kombinačního obvodu. Výstup následujícího stavu je poté přiveden přes klopné obvody na vstup aktuálního stavu, ale toto nijak neovlivňuje návrh kombinační části. Převod do PLA je tak možné provést prostým spojením

vstupního vektoru s vektorem aktuálního stavu a vektoru následujícího stavu s výstupním vektorem. V tabulce 16 je uveden příklad převodu pro benchmark mc.kiss2.

Tabulka 16 - příklad převodu z formátu KISS2 do formátu PLA

automat ve formátu KISS2 (mc.0.kiss2)	***popis kombinační části sekvenčního obvodu ve formátu PLA (mc.0.pla)***
.i 3 .o 5 .p 10 .s 4 .r 00 0-- 00 00 00010 -0- 00 00 00010 11- 00 10 10010 --0 10 10 00110 --1 10 01 10110 10- 01 01 01000 0-- 01 11 11000 -1- 01 11 11000 --0 11 11 01001 --1 11 00 11001	.i 5 .o 7 .ilb i0 i1 i2 st0 st1 .ob stn0 stn1 o0 o1 o2 o3 o4 .type fr .p 10 0--00 0000010 -0-00 0000010 11-00 1010010 --010 1000110 --110 0110110 10-01 0101000 0--01 1111000 -1-01 1111000 --011 1101001 --111 0011001 .e

Takto vytvořený formát PLA obsahuje pouze popis kombinační části automatu. Neobsahuje informaci o počtu bitů původního stavového vektoru a o startovním stavu původního obvodu. Používá se pro minimalizaci kombinační části automatu.

4.4 Převod PLA do VHDL

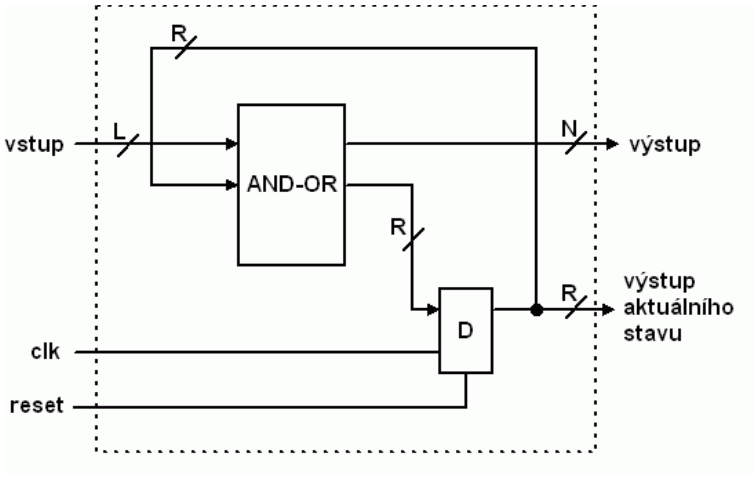
Formát PLA byl popsán v kapitole 4.1.2. V kapitole 4.3 jsme popsali postup převodu formátu KISS2 do formátu PLA. PLA neobsahuje informace o tom, které vstupy a výstupy představují zpětnou vazbu. Tuto informaci je nutné získat jinak, například z původního souboru KISS2.

Převod PLA do VHDL obsahuje dva kroky, převod popisu kombinační části z původního PLA a vytvoření zpětné vazby se synchronními klopnými obvody s možností nastavení startovního stavu.

Kombinační část je ve VHDL tvořena paralelními příkazy. PLA obsahuje pravdivostní tabulku pro L vstupů a N výstupů. Každý řádek levého sloupce tabulky představuje jeden minterm. Každému mintermu odpovídá jeden vnitřní signál a paralelní příkaz. Každý výstupní bit je vyhodnocen jako logický součet signálů příslušných mintermů.

Zpětná vazba je tvořena jedním procesem, který reaguje na změny na hodinovém signálu a na signálu RESET. V případě, kdy má signál RESET hodnotu logická 1, je na signály aktuálního stavu přiveden startovní vektor. V případě, kdy má signál RESET hodnotu log. 0, je při náběžné hraně na hodinovém signálu převedena hodnota ze signálů následujícího stavu na signály aktuálního stavu.

Vstupem obvodu jsou primární vstupy původního sekvenčního obvodu a dva řídící signály, výstupem jsou primární výstupy původního obvodu. Výstup lze rozšířit o signály aktuálního stavu. Struktura vytvořeného obvodu je na obrázku 15.



Obrázek 15 - Struktura obvodu po převodu z PLA do VHDL

4.5 Generování testu pro sekvenční obvody

Testovací posloupnost pro sekvenční obvody zpravidla obsahuje kromě testovacích vektorů i nastavovací vektory, pomocí kterých se nastavuje požadovaný vnitřní stav obvodu. V této práci se zabýváme samočinně testovanými obvody, u kterých je množina testovacích vektorů tvořena množinou přípustných slov obvodu.

V případě testu pro skutečný obvod by se jednalo o nalezení Eulerovského sledu (případně tahu). Tato úloha je nazývána také úlohou „čínského pošťáka“ (viz [22]). Patří do třídy P. Pro její řešení lze použít Edmonsonův algoritmus [22].

V našem případě pro otestování obvodu používáme simulátor FE-SIM (viz kapitola 4.6). Simulátor FE-SIM umožňuje návrat do předem uloženého stavu. Test obvodu bude potom probíhat tak, že se otestují všechny přechody z aktuálního stavu (po provedení přechodu se provede návrat zpět do aktuálního stavu) a poté se teprve provede přechod do dalšího stavu. Po navštívení všech dostupných stavů je test hotov.

Množinu testovacích vektorů pro každý stav získáme z popisu obvodu (jedná se o množinu přípustných vstupních slov pro daný stav). Zaměříme se na nalezení posloupnosti nastavovacích vektorů, která umožní navštívení všech dostupných uzlů. Vzhledem k tomu, že časová náročnost přechodu mezi stavy je výrazně menší než časová náročnost testu jednotlivých stavů, se nebudeme snažit o nalezení optimálního řešení, ale libovolného suboptimálního řešení. Při výpočtu budeme používat množinu dostupných nenavštívených uzlů M. Písmenem a budeme označovat aktuální uzel a písmenem b následující uzel. Písmenem P označíme hledanou posloupnost uzlů. Postup je následující:

1. nalezení nejkratších cest mezi všemi páry uzlů (Floydův-Warshallův algoritmus, viz [23])

2. vyřazení všech uzlů nedostupných ze startovního stavu. Tím získáme množinu M dostupných nenavštívených uzlů.

3. nastavení obvodu do startovního stavu s, $a = s$, $M = M \setminus \{s\}$. Uzel s přidáme do zatím prázdné posloupnosti P.

4. vybrání následujícího uzlu b. Může nastat jedna ze tří možností

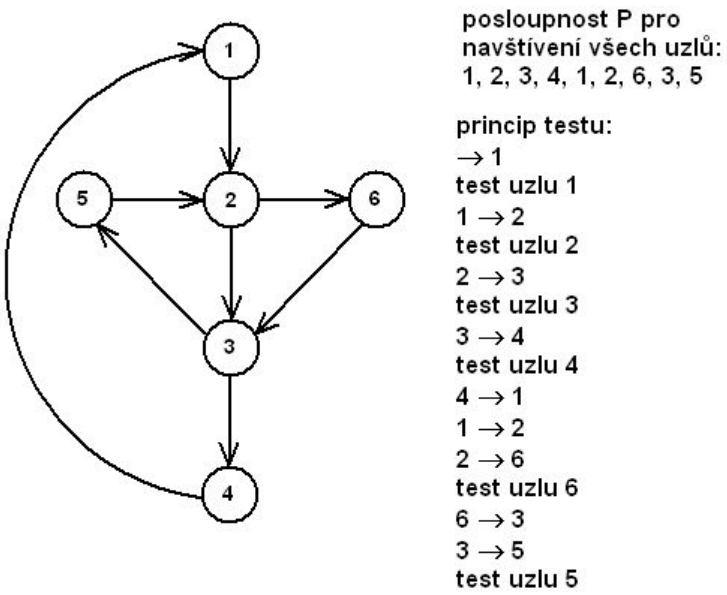
   - z uzlu a je přímo dostupný (tzn. je sousedem uzlu a) alespoň jeden nenavštívený uzel. Následujícím uzlem b se stane libovolný z těchto uzlů.

   - uzel a nesousedí s žádným dosud nenavštíveným uzlem. Prohledáním množiny M dostupných uzlů získáme množinu M_a nenavštívených uzlů nepřímo dostupných z a. Vybereme c z množiny M_a a s pomocí výsledků z bodu 1 nalezneme cestu mezi uzlem a a uzlem c. Jestliže tato cesta obsahuje pouze již navštívené uzly, je přidána k vytvářené posloupnosti P a uzel c se stává následujícím uzlem, $b = c$. V případě, že cesta obsahuje alespoň jeden dosud nenavštívený uzel, vybereme první nenavštívený uzel c' na této cestě. K posloupnosti P je přidána cesta mezi uzlem a a uzlem c' a následujícím uzlem se stává uzel c', $b = c'$.

   - uzel a nesousedí s žádným dosud nenavštíveným uzlem a prohledáním množiny M zjistíme, že žádný dosud nenavštívený uzel není nepřímo dostupný z a. V takovém případě přidáme do posloupnosti P přechod do startovního stavu s (tento přechod odpovídá resetu obvodu). Vybereme libovolný uzel c z množiny M a nalezneme nejkratší cestu mezi uzlem s a uzlem c. Další postup je shodný s předchozím případem s tím rozdílem, že nepracujeme s cestou mezi uzly a a c, ale s cestou mezi uzly s a c.

5. Uzel b se stane aktuálním uzlem, $a = b$, a vyjmeme ho z množiny nenavštívených uzlů $M = M \setminus \{s\}$. Opakujeme krok 4, dokud není množina M prázdná. Posloupnost P obsahuje sled uzlů, který obsahuje každý uzel grafu minimálně jednou.

Testovací posloupnost je založena na posloupnosti nalezené předchozím algoritmem. Protože se v ní libovolný uzel může vyskytnout více než jedenkrát, provádí se test přechodů z uzlu při jeho prvním výskytu v posloupnosti, další výskyty jsou pouze součástí nastavovací posloupnosti pro jiný uzel. Příklad testu je na obrázku 16.



Obrázek 16 - test pro automat

4.6 Simulátor poruch FE-SIM

Dalším krokem bylo nalezení nebo vytvoření vhodného simulátoru s možností vkládání poruch do obvodu. Výhodnější bylo napsání vlastního simulátoru (důvody viz úvod kapitoly 4), který tak bude přesně splňovat naše požadavky. Jako vstupní formát je použit formát EDIF, který přímo popisuje strukturu obvodu v hradlovém poli FPGA a vkládané poruchy tak budou více odpovídat skutečným defektům. Dalším vstupem simulátoru je popis testu ve formátu TST. Simulátor umožňuje simulování poruch jak v kombinačních, tak v sekvenčních obvodech.

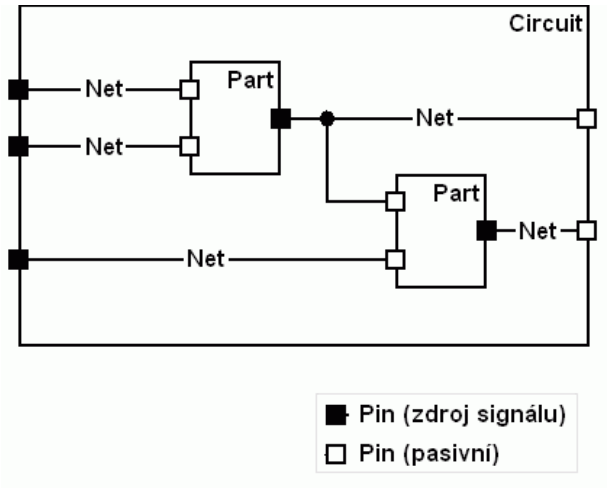
4.6.1 Popis simulátoru

Popis obvodu ve vnitřní formě simulátoru se skládá z objektů čtyř typů: pinů, vodičů (nets), součástek (parts) a obvodu (circuit):

- Pin – objekt pro uložení binární hodnoty. Představuje vstupní nebo výstupní pin součástky nebo celého obvodu. Obsahuje proměnnou s binární hodnotou pinu. Kromě nastavení a čtení této proměnné je pro simulaci užitečná i funkce uložení aktuální hodnoty pinu do pomocné proměnné a možnost pozdějšího obnovení hodnoty pinu (lze tak například uvést obvod do předem uloženého stavu). Umožňuje také zjistit, zdali byla hodnota pinu od posledního čtení změněna, čímž umožňuje zpracování hran signálu (např. hodinového signálu u synchronního klopného obvodu)
- Součástka (Part) – objekt pro výpočet hodnot. Obsahuje množinu vstupních a výstupních pinů (objektů typu Pin). Podle hodnot svých vstupních pinů nastaví hodnoty výstupních pinů.

- Vodič (Net) – objekt pro přenos logických hodnot. Obsahuje jeden zdrojový pin a množinu pasivních pinů. Nejdůležitější metodou je přenos hodnoty ze zdrojového pinu na všechny ostatní piny připojené k vodiči.
- Obvod (Circuit) – Objekt nadřazený všem předchozím objektům. Obsahuje množinu vstupních a výstupních pinů (objektů typu Pin), množinu součástek (objektů typu Part) a množinu vodičů (objektů typu Net). Obsahuje metody pro:
    - vytvoření obvodu – přidávání vstupních a výstupních pinů, přidávání součástek a vodičů a jejich vzájemné propojení
    - výpočet – nastavení vstupů obvodu, výpočet a čtení hodnot na výstupu obvodu
    - simulaci poruch – vytvoření množiny poruch v obvodu, vkládání poruch, uložení vlivu konkrétní poruchy a vytváření statistik simulace poruch.

Schéma jednoduchého obvodu je na obrázku 17.



Obrázek 17 - třídy simulátoru

4.6.2 Průběh simulace

Nejprve podrobněji popíšeme metodu pro výpočet výstupu součástky a metodu pro šíření hodnoty po vodiči.

Výpočet výstupu součástky – metoda vypočte ze vstupů součástky novou hodnotu výstupů a vrací pole výstupních pinů, kterým byla tímto krokem změněna hodnota. Jestliže se hodnota žádného výstupního pinu nezměnila, vrací prázdné pole.

Šíření hodnoty po vodiči – metoda porovná novou hodnotu k distribuci s předchozí hodnotou a jestliže se tyto hodnoty liší, provede přenos nové hodnoty na všechny připojené piny a vrací množinu rodičovských součástek těchto pinů (tzn. součástek, kterým se změnil vstup a u kterých je možné, že se změní i výstup). Výstupní piny nemají rodičovskou součástku, a proto nemohou ovlivnit množinu vracenou touto metodou.

Výpočet výstupu pro jeden vstupní vektor probíhá v následujících krocích:

1. Nastavení hodnot vstupních pinů – na vstupní piny je přiveden nový vektor a hodnota z těchto pinů je distribuována připojenými vodiči. Každý vodič vrací množinu součástek, kterým změnil vstupy. Sjednocením těchto množin získáme množinu A všech součástek se změněným alespoň jedním vstupním pinem.

2. Krok výpočtu
    - vytvoření množiny B, která bude obsahovat součástky se změněnými vstupy.
    - zpracování množiny A. Množina A je postupně procházena. Každá součástka provede nový výpočet a v případě změny některého z jejích výstupních pinů vrací nenulové pole změněných pinů. Pro každý z těchto pinů se provede distribuce nové hodnoty po připojeném vodiči a ovlivněné součástky jsou přidány do množiny B.
    - Pokud je množina B neprázdná (tzn. byl změněn vstup alespoň jedné součástky), je množina A nahrazena množinou B a opakuje se další krok výpočtu. V opačném případě je výpočet ukončen.

3. Přečtení hodnot na výstupních pinech signálu.

Uvedený postup je výhodný v tom, že v každém kroku jsou znovu vyhodnocovány pouze ty součástky, kterým se změnil vstup. Pokud tato změna nezpůsobila změnu výstupu součástky, je výpočet v dané větvi ukončen. Tím může být podstatně urychlena simulace obvodu.

Simulátor FE-SIM neumožňuje simulaci obvodů, které obsahují asynchronní zpětné vazby (tzn. zpětné vazby tvořené pouze vodičem).

Dále popíšeme způsob simulace poruch. Před vložením první poruchy pro aktuální testovací vektor je uložen stav všech pinů. Poruchy jsou vkládány změnou hodnoty pinu. Způsobem výpočtu je zajištěno, že se porucha rozšíří dále po obvodu, případně až na výstupy obvodu. Po uložení výsledku simulace aktuálně vložené poruchy je hodnota všech pinů v obvodu obnovena a postupně jsou vkládány další poruchy z množiny poruch. Po zpracování všech poruch se to samé provede pro všechny další testovací vektory.

4.7 Vyhodnocení výsledku testu a rozdělení poruch

Při testu obvodu (kombinačního i sekvenčního) jsou pro každý testovací vektor postupně vkládány poruchy z množiny poruch a pro každou poruchu zvlášť se uchovává, jak se pro ten který testovací vektor projevila. Po skončení testu obsahuje každá porucha počty (případně seznamy) vektorů rozdělené podle toho, jaký projev poruchy způsobily.

Pro kombinační obvody obsahuje každá porucha tři seznamy:

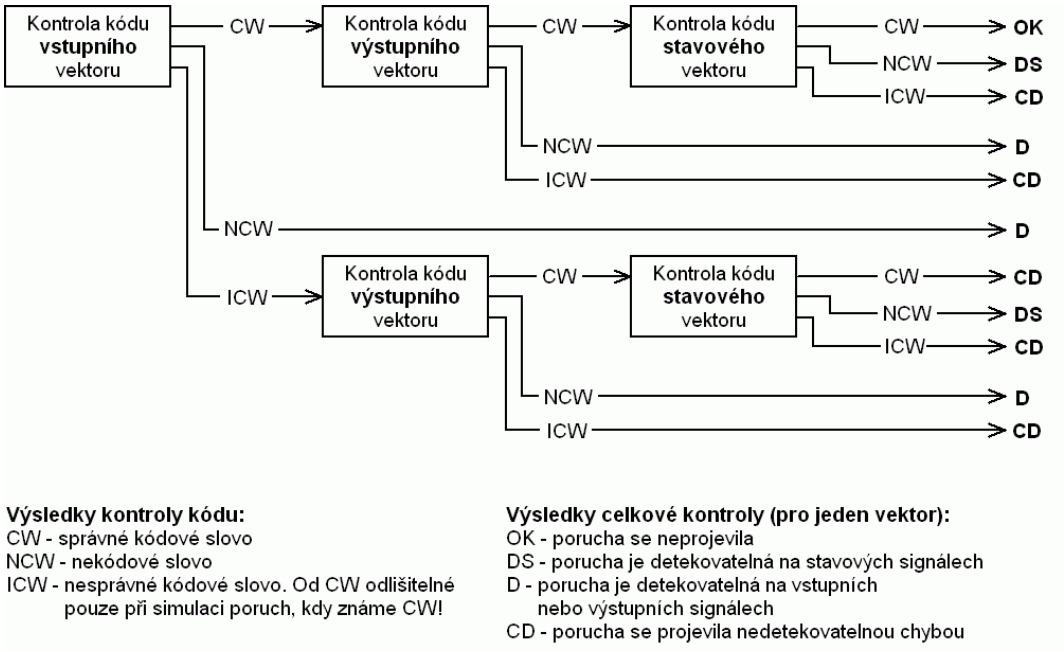
- pasivní vektory - seznam vektorů, pro které se daná porucha nijak neprojevila na výstupu
- testovací vektory - seznam vektorů, pro které se daná porucha projevila nekódovým slovem na výstupu. Způsobují tedy detekovatelnou chybu.
- chybové vektory - seznam vektorů, pro které se daná porucha projevila nesprávným kódovým slovem na výstupu. Tyto vektory tedy způsobují, že se porucha projeví nedetekovatelnou chybou na výstupu.

V celkové statistice můžeme poruchy podle velikosti jednotlivých seznamů rozdělit do čtyř skupin. Rozdělení je v tabulce 17.

Tabulka 17 - rozdělení poruch z hlediska samočinné kontroly

označení poruchy	***popis projevu poruchy a její význam***
A skryté poruchy	Poruchy, které se pro žádný vektor neprojeví. Seznam testovacích vektorů i seznam chybových vektorů je prázdný. Tyto poruchy nemají vliv na bezpečnost obvodu proti poruchám (FS), pro zajištění úplně samočinné kontroly (TSC) by jejich počet měl být nulový.
B detekovatelné poruchy	Poruchy, u kterých seznam testovacích vektorů obsahuje alespoň jeden vektor a seznam chybových vektorů je prázdný. Jestliže se tato porucha projeví, tak vždy pouze nekódovým slovem. V žádném případě se neprojeví nesprávným kódovým slovem. Pro zajištění úplné samočinné kontroly by všechny poruchy měly patřit do této skupiny
C nedetekovatelné poruchy	Poruchy, u kterých seznam chybových vektorů obsahuje alespoň jeden vektor a seznam testovacích vektorů je prázdný. V každém případě, kdy se tato porucha projeví, způsobí nedetekovatelnou chybu na výstupu. Pro zajištění bezpečnosti proti poruchám musí být tato množina prázdná.
D příležitostně detekovatelné i nedetekovatelné poruchy	Poruchy, které mají neprázdný jak seznam testovacích vektorů, tak seznam chybových vektorů. Pro některé vektory se tedy porucha projeví detekovatelnou chybou, pro jiné nedetekovatelnou chybou. Právě kvůli neprázdnému seznamu chybových vektorů by měla být tato skupina poruch také prázdná.

V případě sekvenčních obvodů je vyhodnocování vlivu poruchy složitější. Jestliže je povoleno sdílení prostředků pro výpočet výstupu obvodu a následujícího vnitřního stavu, může se porucha projevit různě na vstupním, stavovém a výstupním vektoru. Kontrola kódu stavového vektoru je prováděna až na výstupech klopných obvodů, případná chyba při výpočtu následujícího stavu se projeví až po hodinovém taktu. Rozdělení vektorů do skupin podle toho, jaké projevení poruchy způsobí, se provádí podle diagramu na obrázku 18. Je nutné si uvědomit, že rozlišení správného a nesprávného kódového slova můžeme provést pouze při simulaci, kdy správné kódové slovo známe.



Obrázek 18 - detekce poruch u sekvenčních obvodů

Při kontrole kódu se tedy používají tři hlídače kódu a pro indikaci chyby stačí, aby jediný z nich detekoval nekódové slovo. Jedinou výjimku tvoří případ, kdy porucha není detekovatelná na vstupu a na výstupu se objeví nesprávné kódové slovo (ICW). Takové projevení poruchy je označeno za nedetekovatelné bez ohledu na výsledek kontroly kódu na stavovém vektoru. Důvodem je už jednou zmiňované zpoždění při kontrole kódu stavového vektoru.

Každá porucha bude obsahovat počty (případně seznamy) čtyř typů vektorů:

- pasivní vektory - vektory, pro které se porucha nijak neprojevila na vstupu, výstupu a ani na stavových signálech obvodu. Na obrázku 18 je výsledek kontroly kódu pro tyto vektory označen symbolem *OK*.
- testovací vektory 1 - seznam vektorů, pro které se porucha projeví detekovatelnou chybou na vstupu a/nebo výstupu obvodu. Na projevu poruchy na stavových signálech nezáleží. Testovací vektory této skupiny způsobí okamžité projevení poruchy. Na obrázku 18 je výsledek kontroly kódu pro tyto vektory označen symbolem *D*.
- testovací vektory 2 - seznam vektorů, pro které je porucha detekovatelná pouze na stavových signálech. Na výstupu obvodu musí být správné kódové slovo. I přes zpožděnou detekci chyby tak nemůže na výstupu nastat nedetekovatelná chyba. Na obrázku 18 je výsledek kontroly kódu pro tyto vektory označen symbolem *DS*.
- chybové vektory - seznam vektorů, které způsobí nedetekovatelnou chybu. Na obrázku 18 je výsledek kontroly kódu pro tyto vektory označen symbolem *CD*.

Rozdělení poruch do skupin podle projevu je stejné jako u kombinačních obvodů, seznam testovacích vektorů je vytvořen spojením seznamu testovacích vektorů 1 a testovacích vektorů 2.

5 Experimenty

V této kapitole budou popsány postupy a výsledky prováděných experimentů. Všechna data a podrobné výsledky jsou k dispozici na přiloženém CD-ROM.

5.1 Porovnání bezpečnostních kódů

V tomto experimentu provedeme porovnání některých systematických bezpečnostních kódů schopných detekovat všechny jednosměrné chyby z hlediska počtu bitů potřebných pro zakódování výstupu obvodu.

Budeme se zabývat Bergrovým kódem a redukovaným kódem M z N. Počet kontrolních bitů m lze u Bergrova kódu vypočítat podle vzorce $m = \lceil \log_2(k+1) \rceil$, kde k je počet informačních bitů. V případě redukovaného kódu M z N závisí počet kontrolních bitů na množině výstupních slov obvodu. Pro některé obvody jsme počet kontrolních bitů pro tento kód zjistili zakódováním výstupu obvodu pomocí námi vytvořeného nástroje.

Výsledné hodnoty pro oba kódy jsou uvedeny v tabulce 18. N je počet bitů nezakódovaného slova, N_{Berger} je počet bitů po zakódování výstupu Bergerovým kódem a $N_{m\,z\,n}$ je počet bitů výstupu při použití redukovaného kódu M z N.

Tabulka 18 - počet bitů kódového slova při zakódování výstupu benchmarků. N je původní počet bitů, další sloupce odpovídají počtu bitů pro Bergerův kód a pro redukovaný kód M z N

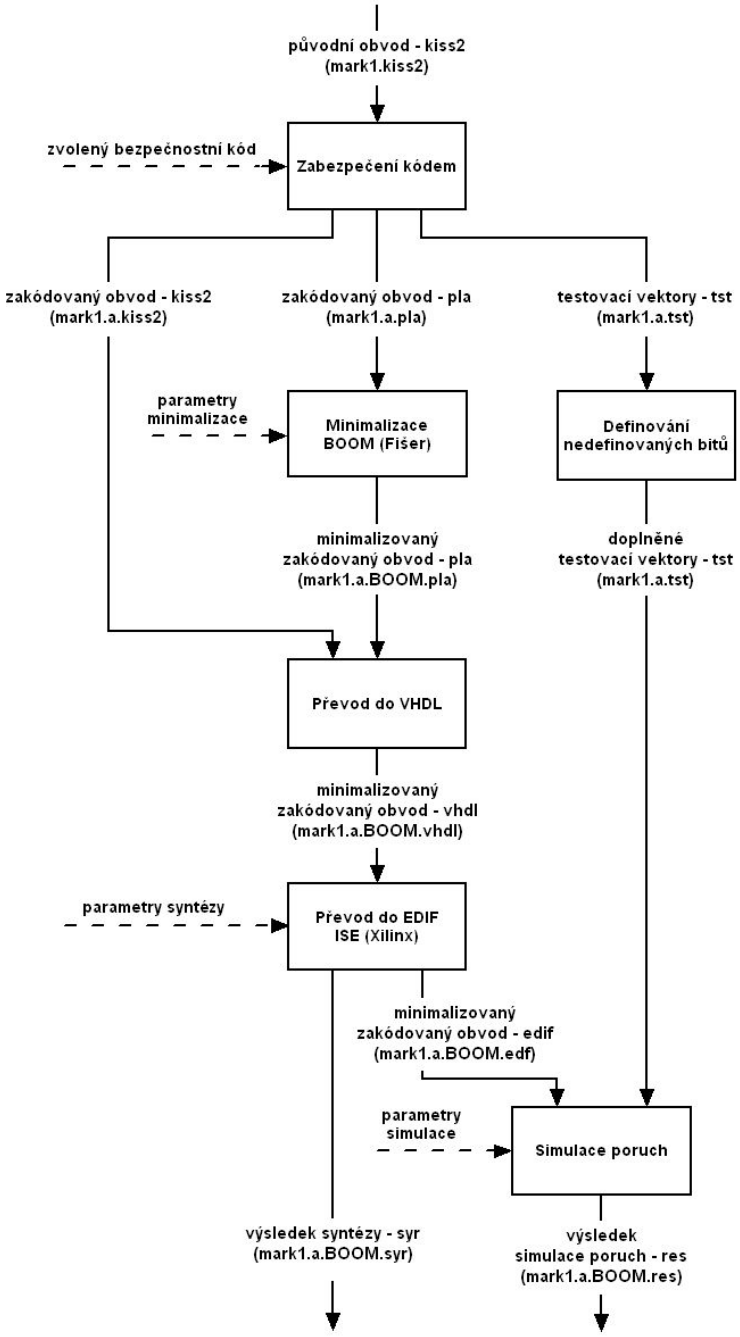
benchmark	***N***	N_{Berger}	$N_{m\,z\,n}$
ex6	8	**12**	14
pma	8	**12**	15
planet	19	**24**	29
s1494	19	**24**	30
ex1	19	**24**	28
scf	56	**62**	66
dk14	5	8	8
s1	6	9	9
cse	7	10	10
ex4	9	13	13
mc	5	8	**7**
mark1	16	21	**17**
s832	19	24	**22**
s820	19	24	**22**

Pro uvažovanou množinu obvodů můžeme oba kódy považovat za rovnocenné, v některých případech byl výhodnější Bergerův kód, v jiných naopak redukovaný kód M z N. Ten je vhodný zejména pro obvody, u kterých má každé výstupní slovo malý počet bitů s hodnotou logická 1.

5.2 Návrh bezpečných obvodů běžným způsobem

V tomto experimentu ukážeme, že pouhé zabezpečení vstupu, stavových signálů a výstupu sekvenčního obvodu bezpečnostním kódem bez přizpůsobení implementace použitému bezpečnostnímu kódu nezajistí bezpečnost obvodu proti poruchám a následkem toho nemůže být takový obvod ani úplně samočinně kontrolovaný. Zjistíme nárůst velikosti obvodu v důsledku zabezpečení a schopnost detekce poruch. Nebudeme se zabývat návrhem hlídače kódu.

Zpracování vstupních dat probíhalo podle diagramu na obrázku 19.



Obrázek 19 - postup zpracování dat v experimentu II

Jako vzorové soubory pro tento test opět posloužily benchmarky ve formátu KISS2. Pro implementaci byl vybrán obvod Virtex II od firmy Xilinx. Aby simulované poruchy v obvodu co nejvíce odpovídaly skutečným poruchám v FPGA, byl pro simulaci poruch vybrán formát EDIF.

Prvním krokem bylo zakódování výstupu, stavových proměnných a případně vstupu obvodu. Byly zvoleny pouze některé kombinace bezpečnostních kódů (viz. tabulka 19). Uvažovali jsme i vliv zakódování vstupů na výsledné vlastnosti obvodu.

Prvním krokem byl kromě popisu automatu získán i soubor s testem, obsahujícím posloupnost vektorů zajišťující otestování všech přechodů automatu. Nedefinované hodnoty ve vektorech testovací posloupnosti byly dodefinovány tak, že každý nedefinovaný bit byl nahrazen dvojicí hodnot log. 1 a log. 0. Předpokládáme, že pravděpodobnost výskytu obou hodnot v nedefinovaném bitu je stejná. Tímto krokem se samozřejmě zvětšil počet testovacích vektorů. Je nutné si uvědomit, že při posuzování úplné samočinné kontroly obvodu i bezpečnosti proti poruchám je množina testovacích vektorů tvořena pouze vektory z množiny přípustných vstupních obvodu. Nelze použít triviální test.

Popis zakódovaného automatu byl minimalizován pomocí nástroje BOOM [24] a převeden do formátu VHDL (viz kapitola 4.4), aby jej bylo možno zpracovat nástrojem pro syntézu obvodů v FPGA. Pro syntézu byl použit nástroj ISE 6.1 od firmy Xilinx. Výsledkem syntézy je soubor EDIF a soubor s parametry takto získaného obvodu.

Posledním krokem byla simulace poruch v obvodu. Použili jsme simulátor poruch FE-SIM (viz kapitola 4.6), umožňující simulaci poruch v obvodu zadaného ve formátu EDIF. Uvažovali jsme poruchy typu t na vstupech a výstupech prvků v obvodu (LUT, buffery, multiplexory a podobně). Těmito poruchami jsou z hlediska bezpečnosti proti poruchám pokryty i poruchy typu SEU v konfigurační paměti LUT a v klopných obvodech v obvodu. Poruchy jsou rozděleny do čtyř skupin, popis jednotlivých skupin je v tabulce 17 v kapitole 4.7.

Dalším cílem tohoto experimentu bylo zjištění velikosti zabezpečeného obvodu. Uvažovali jsme pouze velikost obvodu, nezabývali jsme se velikostí hlídače kódu. Při porovnávání velikosti původního a zabezpečeného obvodu byla jako velikost původního obvodu uvažována velikost menšího ze dvou obvodů, které vzniky zakódováním vnitřního stavu kódem 1 z N a binárním kódem.

Použité nástroje s výjimkou nástroje pro minimalizaci a nástroje pro syntézu obvodu do FPGA byly vyvinuty v rámci této diplomové práce. Jejich podrobnější popis byl uveden v kapitole 4.

Použité formáty souborů s výjimkou formátu pro popis testovacích vektorů a formátu pro uložení výsledku testu patří ke standardním formátům. Popis všech použitých formátů je také v kapitole 4. Jména souborů byla pro zvýšení přehlednosti vytvořena podle následujícího schématu:

jméno souboru = jméno benchmarku[.zvolené kódování][.způsob minimalizace].zkratka formátu

Způsob kódování je označen jedním písmenem podle tabulky 19. Byl použit pouze jeden způsob minimalizace, a to pomocí nástroje BOOM. Takto je označen i v názvu souboru. V diagramu na obrázku 19 je uveden příklad pojmenování souborů pro benchmark mark1.kiss2.

Tabulka 19 - varianty kódování automatu v experimentu II

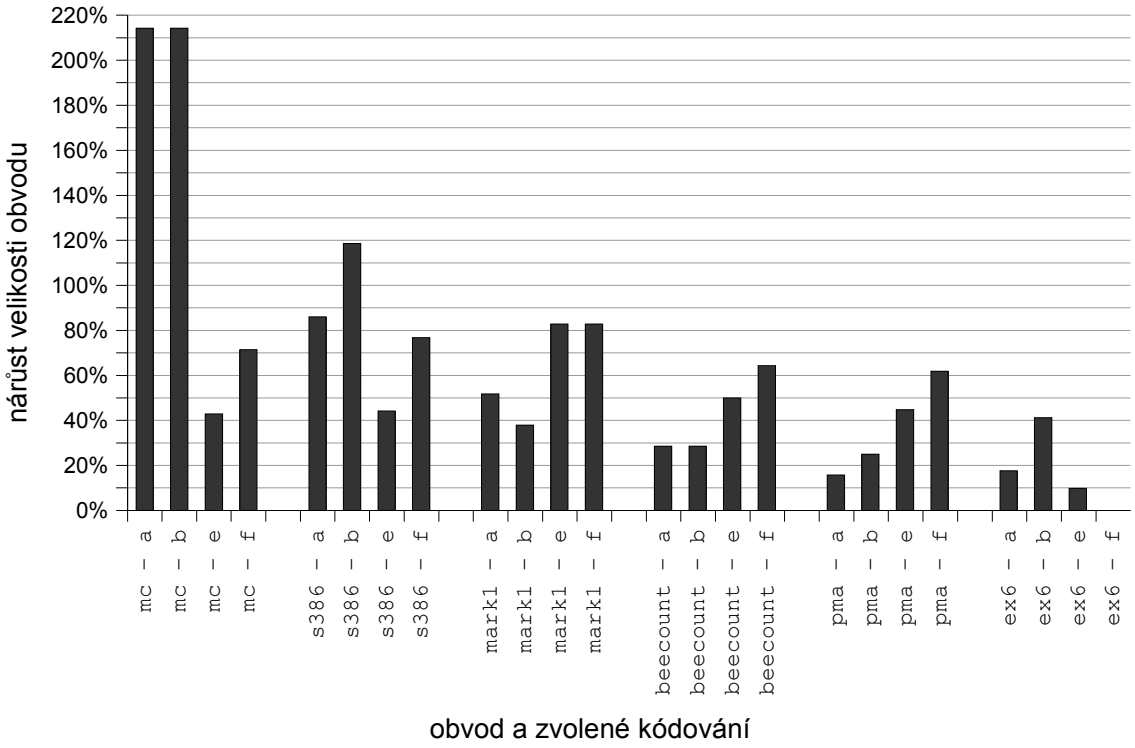
ID	*kód pro vstup*	*kód pro vnitřní stav*	*kód pro výstup*	*poznámka*
0	-	1 z N	-	kódování pro nezabezpečený automat (pouze pro zjištění velikosti originálního obvodu)
1	-	binární	-	
a	-	1 z N	M z N	
b	M z N	1 z N	M z N	
e	-	sudá parita	sudá parita	
f	sudá parita	sudá parita	sudá parita	

V tabulce 20 jsou výsledky pro vybrané benchmarky. V levé části tabulky je jméno benchmarku a varianta kódování (viz tabulka 19). V prostřední části tabulky jsou pro každou variantu kódování uvedeny délky vektorů, velikost obvodu a zvětšení oproti nezabezpečené variantě. V pravé části je rozdělení poruch do kategorií podle tabulky 17.

Tabulka 20 - velikost obvodu a pokrytí poruch pro 6 vybraných benchmarků. Rozdělení poruch použité v pravé části tabulky je popsáno v tabulce 17 v kapitole 4.7

benchmark	varianta kódování	popis kódování	počet vstupních bitů	počet stavových bitů	počet výstupních bitů	počet LUT	nárůst velikosti plochy oproti původnímu obvodu	celkový počet poruch	A – skryté poruchy	A – skryté poruchy [%]	B – detekovatelné poruchy	B – detekovatelné poruchy [%]	C – nedetekovatelné poruchy	C – nedetekovatelné poruchy [%	D – příležitostně detekovatelné i nedetekovatelné poruchy	D – příležitostně detekovatelné i nedetekovatelné poruchy [%]
mc	1	původní obvod	3	2	5	7	x	x	x	x	x	x	x	x	x	x
	a	redukovaný m z n	3	4	7	22	214,3%	322	29	9,0%	272	84,5%	21	6,5%	0	0,0%
	b	redukovaný m z n	6	4	7	22	214,3%	328	32	9,8%	285	86,9%	7	2,1%	4	1,2%
	e	sudá parita	3	3	6	10	42,9%	174	6	3,4%	147	84,5%	21	12,1%	0	0,0%
	f	sudá parita	4	3	6	12	71,4%	194	7	3,6%	172	88,7%	15	7,7%	0	0,0%
s386	0	původní obvod	7	6	7	43	x	x	x	x	x	x	x	x	x	x
	a	redukovaný m z n	7	13	10	80	86,0%	1018	30	2,9%	943	92,6%	45	4,4%	0	0,0%
	b	redukovaný m z n	14	13	10	94	118,6%	1176	41	3,5%	1092	92,9%	7	0,6%	36	3,1%
	e	sudá parita	7	7	8	62	44,2%	746	19	2,5%	529	70,9%	115	15,4%	83	11,1%
	f	sudá parita	8	7	8	76	76,7%	952	63	6,6%	830	87,2%	31	0,0%	28	2,9%
mark1	1	původní obvod	5	4	16	29	x	x	x	x	x	x	x	x	x	x
	a	redukovaný m z n	5	15	18	44	51,7%	678	29	4,3%	616	90,9%	33	4,9%	0	0,0%
	b	redukovaný m z n	10	15	18	40	37,9%	672	19	2,8%	630	93,8%	3	0,4%	20	3,0%
	e	sudá parita	5	5	17	53	82,8%	684	59	8,6%	503	73,5%	86	12,6%	36	5,3%
	f	sudá parita	6	5	17	53	82,8%	682	69	10,1%	533	78,2%	63	9,2%	17	2,5%
beecount	1	původní obvod	3	3	4	14	x	x	x	x	x	x	x	x	x	x
	a	redukovaný m z n	3	7	4	18	28,6%	306	10	3,3%	275	89,9%	17	5,6%	4	1,3%
	b	redukovaný m z n	6	8	4	18	28,6%	324	10	3,1%	293	90,4%	3	0,9%	18	5,6%
	e	sudá parita	3	4	5	21	50,0%	292	18	6,2%	253	86,6%	17	5,8%	4	1,4%
	f	sudá parita	4	4	5	23	64,3%	310	15	4,8%	264	85,2%	12	3,9%	19	6,1%
pma	0	původní obvod	8	24	8	76	x	x	x	x	x	x	x	x	x	x
	a	redukovaný m z n	8	24	15	88	15,8%	1214	29	2,4%	1146	94,4%	17	1,4%	22	1,8%
	b	redukovaný m z n	16	24	15	95	25,0%	1314	29	2,2%	1246	94,8%	3	0,2%	36	2,7%
	e	sudá parita	8	6	9	110	44,7%	1236	105	8,5%	826	66,8%	99	8,0%	206	16,7%
	f	sudá parita	9	6	9	123	61,8%	1374	142	10,3%	1064	77,4%	36	2,6%	132	9,6%
ex6	0	původní obvod	5	8	8	51	x	x	x	x	x	x	x	x	x	x
	a	redukovaný m z n	5	8	14	60	17,6%	790	30	3,8%	725	91,8%	24	3,0%	11	1,4%
	b	redukovaný m z n	10	8	14	72	41,2%	876	54	6,2%	794	90,6%	4	0,5%	24	2,7%
	e	sudá parita	5	4	9	56	9,8%	670	25	3,7%	407	60,7%	143	21,3%	95	14,2%
	f	sudá parita	6	4	9	48	-5,9%	606	23	3,8%	388	64,0%	110	18,2%	85	14,0%

Nejprve se budeme zabývat nárůstem velikosti obvodu způsobené zabezpečením. Na obrázku 20 je odpovídající graf. Udávaná velikost nezahrnuje hlídač kódu.

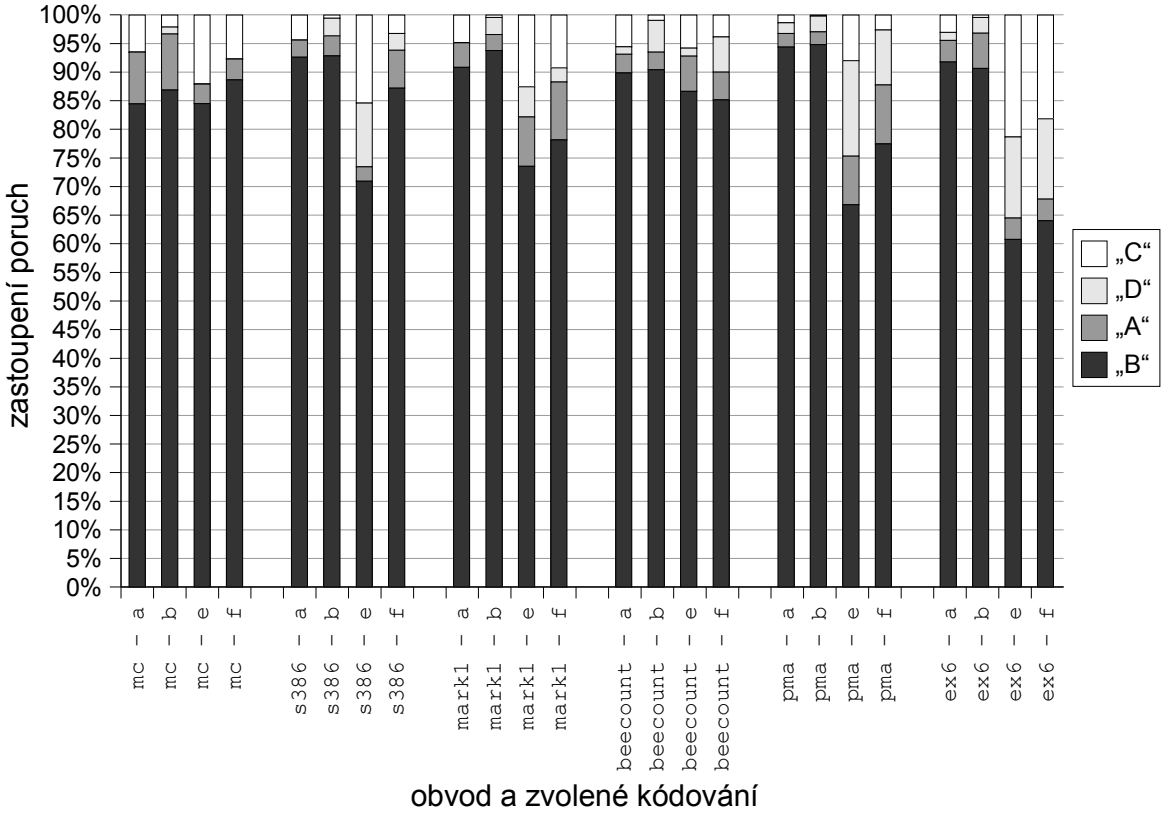


Obrázek 20 - nárůst velikosti obvodu. Vstupní hodnoty pro tento graf jsou uvedeny v tabulce 20.

V případě obvodu mc.kiss2 a kódu M z N (varianta a a b) byl relativní nárůst způsoben velmi malou velikostí původního obvodu (malý počet vstupů a výstupů a malý počet stavů). V ostatních případech dosahoval průměrný nárůst velikosti obvodu hodnot kolem 50% pro oba uvažované kódy. Pro přesnější porovnání by bylo nutné zpracovat větší počet obvodů.

Dalším cílem tohoto experimentu bylo zjištění pokrytí poruch daným bezpečnostním kódem. Na obrázku 21 je graf zobrazující zastoupení poruch v jednotlivých skupinách (viz tabulka 17). Jak již bylo uvedeno výše, poruchy ze skupiny označovaná jako 'B' jsou detekovatelné a nezpůsobují nesprávná kódová slova. Poruchy ze skupiny 'A' jsou skryté poruchy, nemohou mít vliv na bezpečnost obvodu proti poruchám, ale jestliže tato skupina není prázdná, nemůže být příslušný obvod samočinně testovaný. Poruchy ze skupin 'D' a 'C' způsobují nesprávná kódová slova a pokud jsou tyto skupiny neprázdné, nemůže být obvod bezpečný proti poruchám.

Průměrný počet detekovatelných poruch (skupina 'B') pro kód M z N je 91% a pro sudou paritu 77%. V případě, že bychom požadovali obvod bezpečný proti poruchám, můžeme k tomuto číslu přičíst i počet skrytých poruch. Pro kód M z N tvoří skupiny poruch 'A' a 'B' průměrně 96% všech poruch, v případě sudé parity pak 83% všech poruch.



Obrázek 21 - poměrné zastoupení poruch pro různé obvody a různé kódování. Popis rozdělení poruch je v tabulce 17. Vstupní hodnoty pro tento graf jsou uvedenyv tabulce 20.

Kód M z N zajišťoval větší zabezpečení proti poruchám. To je způsobeno tím, že počet různých kódových slov pro vektor pevné délky je pro kód M z N nižší než pro sudou paritu, a tím je nižší i pravděpodobnost, že kódové slovo bude následkem poruchy změněno na jiné, nesprávné kódové slovo. Kód M z N proto považujeme jako vhodnější kód pro zabezpečení obvodu.

Zabezpečení vstupu obvodu vedlo ve všech případech ke zmenšení počtu nedetekovatelných poruch.

Ve všech uvažovaných příkladech ale existuje určité množství poruch, které mohou způsobit nesprávné kódové slovo, a tím nedetekovatelnou chybu na výstupu obvodu. Jednoduchý postup spočívající v zakódování vektorů a implementaci bez ohledu na vlastnosti použitého kódu tak nevede k vytvoření obvodu bezpečného proti poruchám.

5.3 Návrh bezpečných obvodů zajišťující jednosměrné projevení poruch

V tomto experimentu ověříme vlastnosti obvodu implementovaného podle popisu PLA^U (tento formát je popsán v kapitole 3.2). Obvod by podle definice měl být bezpečný proti poruchám. Pokusíme se zjistit, kolik poruch je detekovatelných a jestli takovýto obvod může být samočinně testovaný.

Způsob zpracování dat je podobný postupu z předchozího experimentu. Místo minimalizace programem BOOM [24] byl proveden převod formátu PLA do PLA^U. Ostatní fáze experimentu jsou shodné.

Stejně jako v předchozím experimentu jsme uvažovali poruchy typu t na vstupech a výstupech prvků v obvodu (LUT, buffery, multiplexory a podobně). Neuvažujeme poruchy na hodinovém signálu a poruchy na primárním vstupu pro inicializaci klopných obvodů (asynchronní reset).

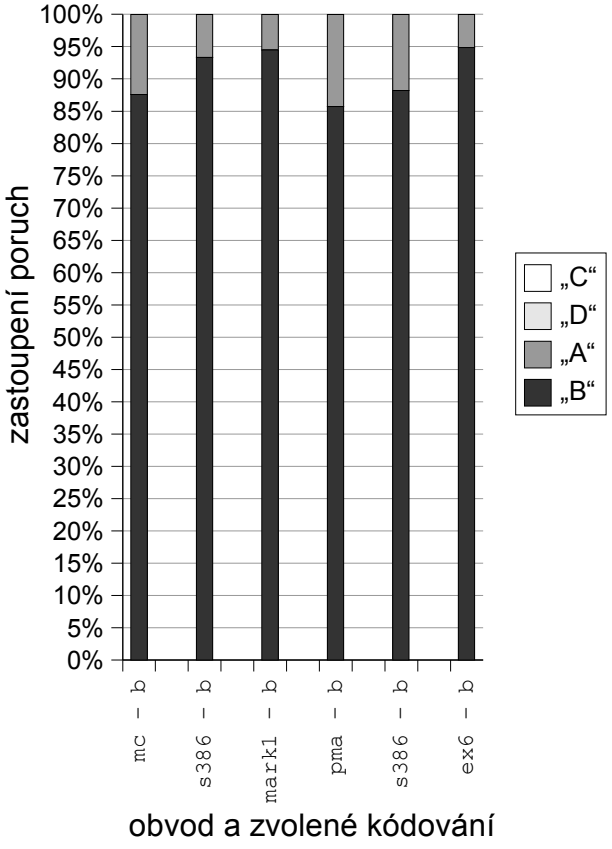
V tabulce 21 jsou výsledky pro vybrané obvody. Pro možnost porovnání jsme zvolili stejné obvody jako v předchozím experimentu.

Tabulka 21 - velikost obvodu a pokrytí poruch pro 6 vybraných benchmarků. Způsob návrhu odpovídá způsobu popsaném v kapitole 3.2 Rozdělení poruch použité v pravé části tabulky je popsáno v tabulce 17 v kapitole 4.7

benchmark	varianta kódování	popis kódování	počet vstupních bitů	počet stavových bitů	počet výstupních bitů	počet LUT	nárůst velikosti plochy oproti původnímu obvodu	celkový počet poruch	A – skryté poruchy	A – skryté poruchy [%]	B – detekovatelné poruchy	B – detekovatelné poruchy [%]	C – nedetekovatelné poruchy	C – nedetekovatelné poruchy [%	D – příležitostně detekovatelné i nedetekovatelné poruchy	D – příležitostně detekovatelné i nedetekovatelné poruchy [%]
mc	1	původní obvod	3	2	5	7	x	x	x	x	x	x	x	x	x	x
	b	redukovaný m z n	6	4	7	26	271,4%	371	46	12,4%	325	87,6%	0	0%	0%	0%
s386	0	původní obvod	7	6	7	43	x	x	x	x	x	x	x	x	x	x
	b	redukovaný m z n	14	13	10	108	151,2%	1307	87	6,7%	1220	93,3%	0	0%	0%	0%
mark1	1	původní obvod	5	4	16	29	x	x	x	x	x	x	x	x	x	x
	b	redukovaný m z n	10	15	18	89	206,9%	1073	59	5,5%	1014	94,5%	0	0%	0%	0%
beecount	1	původní obvod	3	3	4	14	x	x	x	x	x	x	x	x	x	x
	b	redukovaný m z n	6	8	4	32	128,6%	449	64	14,3%	385	85,7%	0	0%	0%	0%
pma	0	původní obvod	8	24	8	76	x	x	x	x	x	x	x	x	x	x
	b	redukovaný m z n	16	24	15	127	67,1%	1637	193	11,8%	1444	88,2%	0	0%	0%	0%
ex6	0	původní obvod	5	8	8	51	x	x	x	x	x	x	x	x	x	x
	b	redukovaný m z n	10	8	14	63	23,5%	833	43	5,2%	790	94,8%	0	0%	0%	0%

Průměrné zvětšení obvodu je více než 140%. Popis PLA^U není možné minimalizovat nástroji používanými pro minimalizaci PLA (například BOOM). Obvod je bezpečný proti poruchám nezávisle na způsobu implementace. V následujících pracích se pokusíme nalézt takový způsob implementace, aby byla velikost takto zabezpečeného obvodu menší.

V grafu na obrázku 22 je rozdělení poruch do skupin podle jejich projevení. Rozdělení poruch do skupin je shodné s předchozím experimentem a je uvedeno v tabulce 17.



Obrázek 22 - Poměr zastoupení poruch pro různé obvody. (viz tabulka 17).Vstupní hodnoty pro tento graf jsou uvedeny v tabulce 21.

Přibližně 91% poruch v obvodu je detekovatelných. Zbytek poruch je skrytých, to znamená, že tyto poruchy se v žádném případě neprojeví. Množina skrytých poruch je nenulová pravděpodobně kvůli omezené množině vstupních vektorů obvodu. Kvůli těmto skrytým poruchám nemůže být obvod samočinně testovatelný.

Takto implementovaný obvod je tedy bezpečný proti uvažovaným poruchám. Není samočinně testovaný, a proto není ani úplně samočinně kontrolovaný.

6 Závěr

Cílem práce bylo prostudovat publikované způsoby návrhu úplně samočinně kontrolovaných sekvenčních obvodů v FPGA, vyzkoušet možnost realizace TSC obvodů v FPGA a vlastnosti takto navržených obvodů ověřit ve vhodném systému simulací.

Pro zajištění úplné samočinné kontroly musí obvod splňovat dvě podmínky – musí být bezpečný proti poruchám a samočinně testovaný. Výsledky jednoho našeho experimentu potvrzují, že tyto podmínky nelze splnit pouhým zakódováním výstupů obvodu bez přizpůsobení implementace použitému kódu. Proto se používají speciální způsoby návrhu.

Prozkoumali jsme a podrobně popsali dva způsoby návrhu sekvenčních TSC obvodů v FPGA. První způsob je označován jako MD architektura a je vhodný pro automaty, které mají malý počet různých výstupních slov. Hlídač obvodu netestuje, jestli je na výstupu kódové slovo, ale jestli je na výstupu správné výstupní slovo. Rozborem návrhu jsme ověřili, že obvod podle MD architektury je bezpečný proti poruchám typu t (s výjimkou poruch na primárních vstupech) a proti poruchám SEU v LUT a ve stavovém registru. Dále jsme ověřili, že je samočinně testovaný pro všechny poruchy typu t s výjimkou poruch na primárních vstupech a několika dalších poruch v hlídači. Obvod podle MD architektury tedy není úplně samočinně kontrolovaný (tzn. samočinně testovaný pro 100% poruch). Je ale samočinně kontrolovaný pro většinu poruch typu t. Zabezpečený obvod byl průměrně o 89% větší než nezabezpečená verze. Nevýhodou MD-architektury je složitý způsob implementace.

Druhý způsob návrhu používá detekční kódy pro zabezpečení vstupu, výstupu a aktuálního stavu obvodu. Původní popis obvodu je změněn na popis obsahující pouze monotónní funkce. Pro zabezpečení je použit redukovaný kód M z N (jedná se o systematický kód M z N). Tento kód je schopný detekovat všechny jednosměrné chyby, a tak je zaručena bezpečnost obvodu proti poruchám typu t, proti poruchám SEU v LUT a proti poruchám SEU ve stavovém registru. Toto tvrzení jsme pro šest benchmarků ověřili simulací. Schopnost samočinného testování závisí na množině vstupních slov obvodu a na způsobu implementace obvodu. Pro vybranou skupinu benchmarků se průměrně 91% poruch typu t projevilo detekovatelnou chybou. Ostatní poruchy se neprojevily (Obvod ale je proti nim bezpečný.) Obvody navržené tímto způsobem tak sice nebyly úplně samočinně kontrolované (tzn. samočinně kontrolované pro všechny poruchy), ale byly samočinně kontrolované průměrně pro 91% poruch. Průměrný nárůst velikosti obvodu byl 140% velikosti nezabezpečeného obvodu (hodnoty se pohybovali v rozmezí od 24% do 270%).

Pro ověřování vlastností navržených obvodů jsme vyvinuli vlastní simulátor poruch – FE-SIM. Simulátor FE-SIM umožňuje simulaci poruch typu t jak v sekvenčních, tak v kombinačních obvodech. Testované obvody jsou zadávány ve formátu EDIF a chyby jsou vyhodnocovány z hlediska samočinné kontroly. Kromě simulátoru poruch FE-SIM byly vyvinuty některé další nástroje: nástroj pro

zabezpečení automatu zvoleným bezpečnostním kódem (popis automatu je ve formátu KISS2), generátor testů pro sekvenční obvody a nástroje pro překlad z formátu KISS2 do formátu VHDL. Všechny nástroje jsou k dispozici na přiloženém CD-ROM.

Obvod navržený podle jedné nebo druhé metody je bezpečný proti poruchám, ale není samočinně testovaný. Počet detekovatelných poruch (tzn. počet poruch, které se pro alespoň jeden vektor projevily nekódovým slovem) jsme u první metody nezjistili, u druhé dosahoval průměrné hodnoty 91% všech poruch.

Předpokládáme další výzkum v této oblasti. Budeme se zabývat vývojem nástrojů pro implementaci sekvenčních obvodů v FPGA podle MD-architektury, abychom měli přesnější informace o jejích vlastnostech. Dále se budeme zabývat vhodným způsobem minimalizace obvodů podle návrhu využívajícího kódy pro detekci jednosměrných chyb. Zaměříme se také na výzkum metod pro snížení počtu nedetekovatelných poruch. Budeme rozšiřovat funkce simulátoru FE-SIM a vyzkoušíme další modely poruch, které budou lépe odrážet skutečné fyzické defekty v FPGA.

7 Literatura

[1] MATOUŠEK, R.: *Informační systémy*. VUT Brno, 2002.

[2] WANG, S.J.: *Error Control Codes for Fault-Tolerant Systems.* National Chung-Hsing University, 2001

[3] HLAVIČKA, J.: *Diagnostika a spolehlivost*. Praha, Vydavatelství ČVUT, 1998

[4] MITRA, S.: *Diversity techniques for concurrent error detection.* Ph.D. Thesis. Stanford University, 2000.

[5] GREBLICKI, S.; PIESTRAK S. J.: *Design of Totally Self-Checking Code-Disjoint Synchronous Sequential Circuit.* The Third European Dependable Computing Conference on Dependable Computing (EDCC-3), Prague, 1999. pp. 251 - 266

[6] ZENG, Ch; SAXENA N.; MCCLUSKEY E. J.: *Finite state machines synthesis with concurrent error detection.*. Stanford University, 1999

[7] LEVIN, I; SINELNIKOV, V.; KARPOVSKY, M.: *Synthesis of ASM-based Self-Checking Controllers.* Euromicro Symposium on Digital Systems Design (DSD'2001), Warsaw, 2001. pp.87-93

[8] LEVIN, I; SINELNIKOV, V.: *Self-Checking of FPGA-based Control Units.* The 9th Great Lakes Symposium on VLSI, Ann Arbor, 1999. pp.292-295

[9] KARPOVSKY, M; LEVIN, I; SINELNIKOV, V.: *New architecture for sequential machines with self-error detection.* International Conference on New Information Technologies (NITe'2000), Minsk, 2000. pp.87-93

[10] LEVIN, I; KARPOVSKY, M; SINELNIKOV, V.: *Architecture of FPGA-based Concurrent Checking FSM*. The 3rd International Electronic Circuits and Systems Conference (ECS2001), Bratislava, 2001. pp.63-68

[11] LEVIN, I; SINELNIKOV, V; KARPOVSKY, M; OSTANIN, S.: *Sequential Circuits Applicable for detecting different types of faults.* The 8th IEEE International On-Line Testing Workshop (IOLTW2002), Isle of Bendor, 2002. pp.44-48

[12] MATROSOVA, A.; OSTANIN, S.: *Self-Checking FSM Networks Design.* The 4th IEEE International On-line Testing Workshop (IOLTW'98), Capri, 1998. pp.162-166

[13] MATROSOVA, A.; LEVIN, I.; OSTANIN, S.: *Self-checking Synchronous FSM Network Design with Low Overhead*. International Journal of VLSI Design, vol. 11, 2000. pp.47-58

[14] MATROSOVA, A.; GOLOUBEVA, O; NIKITIN, K.; OSTANIN, S.: *Totally Self-checking FSM design based on multilevel synthesis methods and FPGA Implementation.* The 4th International Conference on Computer-Aided Design of Discrete Devices (CAD DD 2001), Minsk, 2001.

[15] LEVIN, I.; OSTROVSKY, V.; OSTANIN, S.; KARPOVSKY, M.: *Self-checking Sequential Circuits with Self-healing Ability.* The 12th Great Lakes Symposium on VLSI (GLSVLSI 2002), New York City, 2002. pp.71-76

[16] http://www.ee.vt.edu/~ha/cadtools/cadtools.html

[17] Berkeley Logic Interchange Format (BLIF), University of California, 1992

[18] http://service.felk.cvut.cz/vlsi/prj/BOOM/pla_c.html

[19] http://www.edif.org/

[20] http://splish.ee.byu.edu/download/edifparser/edif_parser_download.html

[21] Tseng, C; Siewiorek D.P.: *Allocation with Clique partitioning.* 1999.

[22] Palúch, S: *Skripta z teorie grafov.* (http://frcatel.fri.utc.sk/users/paluch/), Žilinská Univerzita

[23] Kolář, J: *Teoretická informatika*, Praha, Česká informatická společnost, 2000.

[24] http://service.felk.cvut.cz/vlsi/prj/BOOM/

Příloha A – obsah CD

Adresář	*Obsah*
/benchmarky	Benchmarky ve formátu KISS2.
/experimenty	Data získaná při jednotlivých experimentech a podrobné výsledky experimentů.
/literatura	Použitá literatura (články).
/program – dokumentace	Uživatelské příručky k našim nástrojům.
/program – tridy	Přeložené třídy připravené k použití.
/projekt/doc	Dokumentace ke zdrojovým kódům. Zde jsou podrobně popsány jednotlivé balíky a třídy a jejich metody.
/projekt/src	Zdrojové kódy tříd vyvinutých v rámci této diplomové práce.
/text	Dokumentace k diplomové práci (tento text).