Automatizace návrhu systémů odolných proti poruchám pomocí vysokoúrovňové syntézy

Jakub Lojda
2. ročník, prezenční studium,
Školitel: Doc. Ing. Zdeněk Kotásek, CSc.

Fakulta informačních technologií Vysokého učení technického v Brně
Božetěchova 1/2, 612 66 Brno, Czech Republic
Email: {ilojda, kotasek}@fit.vutbr.cz

***Abstrakt*—S vyšší úrovní integrace přichází výzva maximálně využít dostupnou kapacitu na čipu. Efektivní využití zdrojů je námětem pro vznik nových metod návrhu číslicových obvodů. Jednou z těchto metod je tzv. *vysokoúrovňová syntéza* (HLS), která je využitelná ve spojení s hradlovými poli *Field Programmable Gate Array* (FPGA). Obecným cílem našeho výzkumu je nalézt metodu automatického návrhu systémů odolných proti poruchám. Tento článek shrnuje první kroky výzkumu, které se zabývají možností vkládání redundance s pomocí HLS a zejména vyhodnocením tohoto přístupu.**

***Klíčová slova*—Automatizace návrhu, HLS, vysokoúrovňová syntéza, Catapult C, odolnost proti poruchám, systém odolný proti poruchám.**

I. Úvod

Vyšší úroveň integrace vede na možnost realizovat složitější obvody uvnitř čipů, pro návrháře je k dispozici více zdrojů, které je ale stále těžší efektivně využít. Efektivní využití těchto zdrojů je námětem pro vznik nových metod návrhu obvodů. Jednou z těchto metod je tzv. *vysokoúrovňová syntéza*, angl. *High Level Synthesis* (HLS). K realizaci výsledných obvodů je pak často využívána technologie hradlových polí FPGA. U těchto obvodů však vede rostoucí úroveň integrace na čipu ke zvyšování náchylnosti k tzv. jevům *Single Event Upset* (SEU). Obecným cílem našeho výzkumu je nalézt metodiku pro automatizovanou volbu zajištění *odolnosti proti poruchám* (OPP), která by byla schopna pokrýt jak nové metody návrhu (HLS), tak konvenční přístupy.

Tento článek popisuje konkrétní snahy o zavádění OPP do systémů tvořených pomocí HLS. Text je organizován do šesti částí, Sekce II a Sekce III se věnují souvisejícím výzkumům. Sekce IV popisuje současný stav práce na metodě zavádění spolehlivosti do systémů založených na HLS, Sekce V uvádí vyhodnocení představené metody a Sekce VI uvádí obecné cíle disertace a závěrečné zhodnocení.

II. Související práce: Automatizace návrhu systémů OPP

Autoři příspěvku [7] shrnují náležitosti, které by nástroj pro automatizaci návrhu systémů OPP měl mít. Dále zvažují fakt, že dosavadní pokusy o automatizaci byly vedeny na nižších úrovních abstrakce. Autoři poukazují, že i na vyšších úrovních abstrakce je možné dosáhnout stejně kvalitních výsledků s dodatečnými výhodami v podobě snazší práce návrháře s popisem na úrovni HDL. Další shrnutí tohoto přístupu nabízí hlavní autor předchozí publikace v příspěvku [6]. Hradlová pole FPGA, na která je náš výzkum primárně orientován, je možno dle místa projevu SEU rozdělit do třech úrovní [14]: 1) *vrstva konfigurační paměti*; 2) *vrstva blokových pamětí*; a 3) *vrstva uživatelské logiky*. Proto i následující text je rozdělen tak, aby korespondoval s jednotlivými úrovněmi.

A. Konfigurační paměť

Jedním ze způsobů ošetření konfigurační paměti je tzv. *čištění paměti*, neboli *memory scrubbing*. Autoři příspěvku [13] uvádějí metodu spouštění *scrubbingu* na základě predikce poruch. Jinou variantu představuje tzv. *Frame-Level Redundancy* (FLR) *scrubbing* [19], který je vhodný pro *obnovu* systému založeného na architektuře TMR s jemnou granularitou. Příspěvek [12] uvádí metodu pro snížení *Mean Time To Reapir* (MTTR). Metoda překládá *chybovou signaturu* na startovní rámec, od kterého je při detekci poruchy zahájen *scrubbing*.

B. Paměťové elementy

V [20] je prezentována metodika návrhu číslicových obvodů s ohledem na náchylnost k přechodným poruchám v paměťových elementech. Autoři vyvíjejí nástroj zvaný *FT-PRO*, který tuto metodiku automatizuje. Aplikační poznámka [11] uvádí realizaci *čištění paměti* pro blokové paměti BRAM.

C. Uživatelská logika

Metoda pro modifikaci sekvenčních obvodů tak, aby se staly *samočinně kontrolované*, je prezentována v [10]. Příspěvek [1] se zaměřuje na tvorbu nástroje pro vložení redundance na principu modifikace již existujícího VHDL kódu dle předložených specifikací. Nástroj prohledávající stavový prostor možných konfigurací OPP prezentují autoři příspěvku [16].

III. Související práce: HLS

Následující část textu je věnována popisu aktuálních metod zavádějících OPP do systémů syntetizovaných pomocí HLS, což je také aktuálním předmětem zaměření našeho výzkumu.

A. Datové cesty

V příspěvcích [3] a [5] se autoři zabývají poruchami datových cest, jež se vyskytují přechodně a trvají několik taktů. Heuristika uváděná v [17] umožňuje volit kompromis mezi latencí a mírou redundance výsledného systému. Na zvolené testovací sadě bylo dosaženo 18% až 49% úspory zdrojů při 70% pokrytí poruch a zdvojnásobení přípustné *latence*.

B. Řadič datových cest

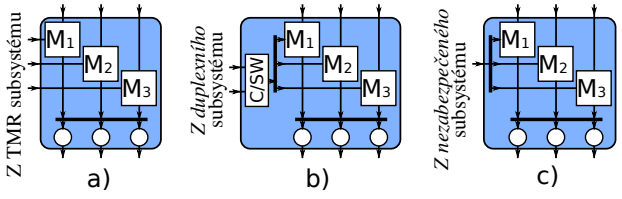
Pro dosažení vysoké míry OPP je mimo zabezpečení datových cest vhodné věnovat dostatečnou pozornost také řadiči, kterým jsou datové cesty ovládány. V případě, že datové cesty podporují detekci přechodných poruch, je možné využít koncept představený v [4].

IV. Současný stav výzkumu

Aktuální snahy spočívají ve vyhodnocení alternativní metody pro vkládání redundance do systémů navržených pomocí HLS. Protože k modifikacím zde dochází na úrovni algoritmu, který je vstupem do vysokoúrovňové syntézy, přístup zachovává výhody, kterými HLS disponuje. Je nutné zdůraznit, že tato metoda nezvažuje zabezpečení řadiče datových cest a příslušejících ostatních prvků, které nejsou přímo ovlivnitelné vstupním popisem algoritmu. První nástin naší metody byl publikován v [8].

Při výzkumu využíváme komerčně dostupný nástroj HLS *Catapult C* [9], který je schopen zpracovat jazyk C (příp. C++) a na svém výstupu poskytuje popis obvodu v jazyce VHDL vyjádřením na úrovni RTL. V algoritmu popsaném v jazyce C++ můžeme identifikovat celkem tři typy modifikací, jsou to: 1) *datové typy* (*úložiště*); 2) *aritmetické a logické operace*; a 3) *příkazy pro řízení toku*. Tato fáze práce je zaměřena na výzkum dosažení OPP pomocí změny *úložišť* a *operací*. Metoda využití nových *datových typů* (DT), které nazýváme *redundantní datové typy* (RDT), bude vyhodnocena na známém principu TMR. Koncept RDT využívá již existující DT (které označujeme jako *originální* DT). Každý RDT vyjadřuje jednu metodu OPP (např. TMR) a může zastřešovat operace nad libovolným *originálním* DT. Tento způsob umožňuje měnit sémantiku operací příslušejících k danému RDT a tak implementovat OPP do těchto operací. Modifikace algoritmu pak spočívá v pouhých záměnách DT za RDT implementující konkrétní metody OPP.

U *binárních* operací může docházet ke komunikaci odlišných subsystémů, na které je aplikován odlišný druh zajištění OPP, může tak docházet k operacím s těmito kombinacemi DT a RDT: **a) intra-DT** operace (RDT vs. RDT zajišťující totožnou metodu OPP); **b) inter-DT** (RDT vs. RDT zajišťující odlišnou metodu OPP); a **c) original-DT** operace (RDT vs. jeho *originální* DT). Příklady možných kombinací pro systém TMR shrnuje Obrázek 1.



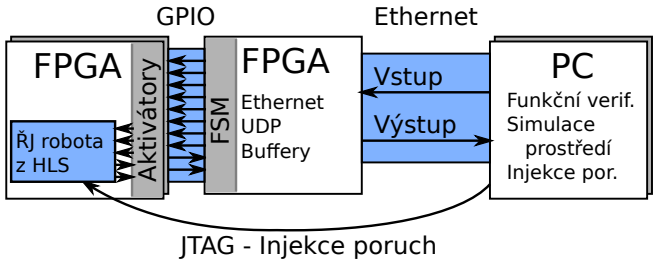
Obrázek 1. Příklad situací, které mohou nastat mezi operacemi v HW pro systém TMR v případě (a) operací *intra-DT*; (b) operací *inter-DT* (v tomto příkladě subsystém TMR komunikuje se *zdvojeným* (*duplexním*) subsystémem); a (c) operace s *originálním* DT.

V. Případová studie: Řadič robota

Vyhodnocení parametrů prezentovaných principů představuje případová studie na řadiči robota pro navigaci v bludišti.

A. Verifikační prostředí s řadičem robota

Pro zkoumání vlastností výsledných implementací bylo využito verifikační prostředí pro vyhodnocení vlivu poruch na zkoumanou jednotku [15]. Verifikační prostředí je vybudováno dle metodiky *Universal Verification Methodology* (UVM) a je zaměřeno na vyhodnocení *elektromechanických* aplikací. Verifikační prostředí je složeno ze dvou částí: 1) elektronického přípravku (*kit* s FPGA, do kterého jsou injektovány poruchy, zde *ML506* založený na FPGA řady *Virtex 5*); 2) mechanického prostředí, které je simulováno v osobním počítači (zde aplikace *Player/Stage* [2]), implementace UVM a injektoru poruch [18], rovněž běžících na PC. Schéma experimentálního prostředí ukazuje Obrázek 2.



Obrázek 2. Struktura experimentálního verifikačního prostředí.

Za pomoci injektoru poruch je možno cíleně vkládat poruchy do využitých bitů *Look-up* tabulek (LUT) a to zároveň pouze do těch, které jsou součástí specifikovaného bloku, tj. zde do verifikovaného obvodu.

B. Volba nastavení parametrů HLS

Protože nástroje HLS umožňují nastavit velké množství parametrů syntézy, bylo zapotřebí identifikovat vhodná nastavení, se kterými bude provedeno vyhodnocení. Jedním z významných parametrů u akceleračních technik v HLS je tzv. *iniciační interval*, jenž značí počet taktů mezi spouštěním jednotlivých cyklů smyčky (může dojít k překryvu běhů). *Úrovní paralelního výpočtu* rozumíme počet souběžně vykonávaných cyklů smyčky. Pro experimenty byla stanovena celkem čtyři různá nastavení HLS: **1) noopt-area** – HLS ve výchozím nastavení, cílem optimalizace byla plocha na čipu; **2) noopt-latency** – HLS ve výchozím nastavení, cílem optimalizace byla *latence*; **3) pipeline1-area** – celý obvod zřetězen při *iniciačním intervalu* jedna, cílem optimalizace plocha na čipu; **4) unroll2-area** – celý obvod rozbalen s *úrovní paralelního výpočtu* dvě, cílem optimalizace plocha na čipu.

C. Vliv nastavení HLS na náchylnost k SEU

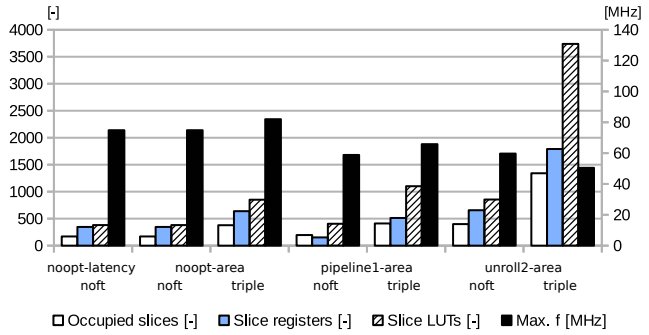
Algoritmus řadiče robota popsaného v jazyce C++ byl syntetizován pomocí HLS se čtyřmi popsanými nastaveními. Takto vznikly čtyři různé implementace řadiče robota, jejichž podrobnější popis z hlediska spotřebovaných zdrojů po syntéze nástrojem *Xilinx ISE* uvádí řádky označené *noft* v Tabulce I. Spotřebované zdroje shrnuje rovněž graf na Obrázku 3. Při bližším ohledání bylo zjištěno, že jednotky *noopt-area* a *noopt-latency* vedly na totožný VHDL kód, proto byla sada nastavení *noopt-latency* z dalších experimentů vyjmuta

Zbývající tři jednotky byly následně vyhodnoceny na náchylnost k poruchám. S každou *řídicí jednotkou* (ŘJ) bylo

Tabulka I
SPOTŘEBOVANÉ ZDROJE PRO KAŽDOU VERZI ŘJ ROBOTA.

Verze		Obsaz. *slices* [−]	*Slice reg.* [−]	*Slice LUTs* [−]	Max. frekv. [MHz]	bity LUT [*b*]
noopt-latency	*noft*	170	346	381	74.85	19392
noopt-area	*noft*	170	346	381	74.85	19392
	triple	378	638	851	82.01	48704
	TMR	546	1038	1143	74.85	58176
pipeline1-area	*noft*	196	152	405	58.82	21952
	triple	411	512	1101	65.81	67264
	TMR	540	456	1215	58.82	65856
unroll2-area	*noft*	399	656	854	59.70	48704
	triple	1341	1791	3738	50.48	224256
	TMR	1224	1968	2562	59.70	146112

provedeno 1000 verifikačních běhů, všechna vyhodnocení probíhala na totožné mapě bludiště. Vyhodnocení probíhalo následovně: 1) ŘJ robota byla uvedena do výchozího stavu; 2) do LUT tabulek obsazených ŘJ robota byla s *uniformním* rozložením pravděpodobnosti vložena jednonásobná porucha; 3) byla sledována schopnost ŘJ nalézt cílovou pozici.



Obrázek 3. Porovnání spotřebovaných zdrojů pro každou z verzí řadiče robota.

Výsledky získané při injekci poruch v tomto kroku popisuje ve sloupcích označených *noft* Tabulka II. Vrchní část tabulky informuje o celkovém stavu elektroniky, tj. zda došlo k odlišnostem mezi vzorovou implementací a implementací s injektovanou poruchou. Spodní část tabulky informuje o počtu případů, kdy robot nenalezl cíl. Tabulka II dále informuje o případech, kdy došlo ke kolizi robota se stěnou příp. kolikrát robot sice dorazil do cíle, ale cesta byla odlišná od vzorové.

Lze konstatovat, že jednonásobné poruchy se nejvíce projevovaly ve variantě *noopt-area*, která nezahrnuje bližší nastavení optimalizací v rámci HLS. Z výsledků však není jisté, zda rozdíly v citlivosti na jevy SEU nejsou způsobeny odlišnou plochou zabranou na čipu, protože počet verifikačních běhů byl shodný pro všechny implementace ŘJ. Z tohoto důvodu by bylo zajímavé jako součást budoucí práce provést rozsáhlejší testování výsledných designů.

Tabulka II
PŘEHLED DOPADU INJEKTOVANÝCH PORUCH PRO KAŽDOU ZE ZVOLENÝCH JEDNOTEK PŘI POČTU 1000 VERIFIKAČNÍCH BĚHŮ.

Dopad poruchy	*noopt-area*		*pipeline1-area*		*unroll2-area*	
	noft	*triple*	*noft*	*triple*	*noft*	*triple*
ŘJ OK [−]	949	982	967	996	979	995
ŘJ selhala [−]	51	18	33	4	21	5
Cíl nenalezen [−]	50	16	32	4	19	4
Kolize [−]	4	2	5	1	4	1
Cíl nalezen jinou cestou [−]	1	2	1	0	2	1

D. Vyhodnocení metody redundantních datových typů

Pro úvodní experiment byla zvolena strategie zabezpečit všechna *úložiště* a *operace* v algoritmu jednotky pomocí metody *ztrojení*. Využitý RDT, nazvaný *triple*, zahrnuje kombinaci *prostorové redundance* v případě replikace HW nástrojem HLS a *časové redundance* v případě optimalizací a mapování některých operací do shodné funkční jednotky. K vyhodnocení byla využita implementace zahrnující ztrojení paměťového elementu a všech operací nad daným RDT. Provedení jedné operace nad RDT zahrnuje spuštění operace nad třemi instancemi *orig*. DT a následné určení výsledku pomocí *majoritní funkce*. Jednotky, v nichž jsou všechna úložiště a operace replikovány na úrovni algoritmu pomocí ruční záměny DT na RDT, byly syntetizovány s vybranými množinami nastavení pro HLS a opět podrobeny popsané metodě vyhodnocení. Tabulka I uvádí rovněž spotřebované zdroje, řádky značené TMR ukazují zdroje jednotek, jež byly syntetizovány ztrojené na úrovni celých ŘJ. Vliv SEU ukazuje Tabulka II.

Závěrem této etapy experimentů lze konstatovat, že nejlepších výsledků dosahuje metoda v kombinaci se *zřetězením*, tj. nastavením HLS *pipeline1-area*, které proto bude využito v následující etapě. Na rozdíl od předchozí etapy nelze v tomto případě pozorovat přímou korelaci velikosti designu s výslednou náchylností k SEU, jako tomu je u jednotek *noft*.

E. Vliv zabezpečení jednotlivých operací

V poslední fázi experimentů byl zkoumán vliv aplikace *ztrojení* jednotlivých úložišť a operací v algoritmu na výslednou celkovou OPP. Úložiště a příslušející operace algoritmu ŘJ robota byly rozděleny do sedmi množin, jež byly označeny celými čísly $1-7$. Přehled zastoupení typů operací v jednotlivých množinách uvádí Tabulka III. Bylo sestaveno sedm korespondujících implementací ŘJ, v každé implementaci byla metoda *ztrojení* aplikována na danou množinu *operací* a *úložišť*. Syntéza byla provedena pomocí sady nastavení *pipeline1-area*, důvodem této volby je vysoká citlivost na zabezpečení prezentovanou metodou. Vyhodnocení je tak možné provést s vyšším rozlišením za konstantního počtu verifikačních běhů v případě, že je podstatný především procentuální rozdíl mezi jednotlivými implementacemi. Sloupec *Ref.* uvádí referenční parametry ŘJ *noft* syntetizované s nastavením *pipeline1-area*.

Dle výsledků 2000 verifikačních běhů lze konstatovat, že částečně zabezpečené jednotky, v nichž byly zabezpečeny množiny operací 2, 5 a 7, dosáhly nejvyšší úrovně OPP. V ostatních případech zabírají ŘJ dokonce menší plochu na čipu, přičemž parametry OPP ŘJ 1, 3 a 6 jsou stále lepší nežli v případě *referenční* jednotky. Věříme, že toto chování souvisí s velikostí řetězených bloků, jež je automaticky volena nástrojem HLS. Je však potřeba dalšího výzkumu pro ověření tohoto předpokladu, protože vzhledem ke snížení spotřebovaných zdrojů je získaná spolehlivost významná.

VI. CÍLE DISERTAČNÍ PRÁCE A ZÁVĚR

Hlavním cílem disertační práce je dospět k obecné metodice pro automatizaci návrhu systémů OPP, pro jeho dosažení byly stanoveny tři hlavní podcíle: 1) vybudovat prostředky pro automatické vkládání redundance do zvoleného jazyka

Tabulka III
EXPERIMENTÁLNÍ VYHODNOCENÍ SPOTŘEBY ZDROJŮ, ZABEZPEČENÝCH OPERACÍ A ÚLOŽIŠŤ A ZÍSKANÉ OPP.

Verze ŘJ	Ref.	1	2	3	4	5	6	7
Bity LUT [b]	21952	17408	55744	12800	15744	47552	15872	35840
Slices [–]	196	147	370	135	165	379	147	250
Selhání [‰]	33.0	27.0	13.5	30.5	37.5	15.5	29.5	17.0
RDT *unární*	0	0	7	22	4	4	2	2
ops. *binární*	0	6	7	32	7	9	5	2
[–] *ternární*	0	0	0	0	0	0	0	0
RDT *inter-DT*	0	0	0	0	0	0	0	0
ops. *intra-DT*	0	0	0	30	4	0	0	2
[–] *orig.-DT*	0	6	14	24	7	13	7	2
Zlepšení OPP [%]	–	18.2	59.1	7.6	-13.6	53.0	10.6	48.5
Prostorová režie [%]	–	-25.0	88.8	-31.1	-15.8	93.4	-25.0	27.6

(jazyků); 2) navrhnout metodu modifikace systému *neodolného* na systém *odolný* – hledat prostor pro zlepšení automatické volby zabezpečení ve vztahu k vlastnostem dané komponenty (příp. části algoritmu) při zohlednění optimalizačních parametrů; 3) zobecnit navržené postupy, aby byly použitelné pro různé jazyky využívané k popisu obvodové realizace.

V rámci podcíle 1) byla zvolena varianta HLS v kombinaci s popisem algoritmu v jazyce C++. Snahou tohoto přístupu je pochopit principy automatizace návrhu systémů OPP na jednodušších případech a poté snaha tyto principy zobecnit a definovat dostatečně obecný aparát, který by umožnil aplikaci těchto metodik automatizovat. K tomuto účelu byla navržena metoda vkládání informační redundance a záložních technických prostředků s využitím speciálně navržených RDT. Podúkol 2) má za cíl využít prostředky pro automatické vkládání redundance za účelem automatizace návrhu. Tyto prostředky umožní částečně abstrahovat metody OPP od jejich konkrétní aplikace. Metoda tedy bude schopna pracovat se systémem již na vyšší úrovni abstrakce, tj. bude rozhodovat na který blok (funkci) v systému aplikovat kterou metodu OPP. Výstupem analýzy bude rovněž granularita aplikace.

Z vyhodnocení metody lze učinit následující závěry: 1) použití popisované metody vkládání OPP se nejvíce projevilo u nastavení *pipeline1-area*; 2) odlišné důležitosti jednotlivých operací v algoritmu lze identifikovat aplikací OPP na tyto operace a následným vyhodnocením celkové spolehlivosti. Jako část budoucí práce by mohlo být zajímavé ověřit důvod fenoménu, jenž se ve zřetězeném designu projevuje u některých operací snížením spotřebovaných zdrojů za současného zvýšení spolehlivosti.

PODĚKOVÁNÍ

Tato práce byla podporována Ministerstvem školství, mládeže a tělovýchovy z Národního programu udržitelnosti (NPU II); projektu IT4Innovations excellence in science – LQ1602. Tato činnost byla rovněž podporována projekty řešenými na VUT v Brně pod číslem FIT-S-14-2297 a ARTEMIS JU pod grantem číslo 641439 (ALMARVI).

REFERENCE

[1] Entrena, L.; Lopez, C.; Olias, E.: Automatic insertion of fault-tolerant structures at the RT level. *Proceedings Seventh International On-Line Testing Workshop*, 2001, s. 48–50, doi:10.1109/OLT.2001.937817.

[2] Gerkey, B.; Vaughan, R.; Howard, A.; aj.: The Player/Stage Project. *přístupné z http://playerstage.sourceforge.net*, 2003.

[3] Inoue, T.; Henmi, H.; Yoshikawa, Y.; aj.: High-level synthesis for multi-cycle transient fault tolerant datapaths. *2011 IEEE 17th International On-Line Testing Symposium*, červenec 2011, ISSN 1942-9398, s. 13–18, doi:10.1109/IOLTS.2011.5993804.

[4] Iwagaki, T.; Ishimori, Y.; Ichihara, H.; aj.: Designing area-efficient controllers for multi-cycle transient fault tolerant systems. *2015 20th IEEE European Test Symposium (ETS)*, květen 2015, ISSN 1530-1877, s. 1–2, doi:10.1109/ETS.2015.7138742.

[5] Iwagaki, T.; Nakaso, T.; Ohkubo, R.; aj.: Scheduling algorithm in datapath synthesis for long duration transient fault tolerance. *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, říjen 2014, ISSN 1550-5774, s. 128–133, doi:10.1109/DFT.2014.6962062.

[6] Leveugle, R.: Automatic modifications of high level VHDL descriptions for fault detection or tolerance. *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, 2002, ISSN 1530-1591, s. 837–841, doi:10.1109/DATE.2002.998396.

[7] Leveugle, R.; Cercueil, R.: High level modifications of VHDL descriptions for on-line test or fault tolerance. *Proceedings 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2001, ISSN 1550-5774, s. 84–91, doi:10.1109/DFTVS.2001.966756.

[8] Lojda, J.; Podivínský, J.; Krčma, M.; aj.: HLS-based Fault Tolerance Approach for SRAM-based FPGAs. *Proceedings of the 2016 International Conference on Field Programmable Technology*, IEEE Computer Society, 2016, ISBN 978-1-5090-5602-6, s. 297–298.

[9] Mentor Graphics: Catapult C/C++/SystemC HLS - Mentor Graphics. `https://www.mentor.com/hls-lp/catapult-high-level-synthesis/c-systemc-hls`, navštíveno: 2016-12-22.

[10] Metra, C.; Francescantonio, S. D.; Omana, M.: Automatic modification of sequential circuits for self-checking implementation. *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, listopad 2003, ISSN 1550-5774, s. 417–424, doi:10.1109/DFTVS.2003.1250139.

[11] Miller, G.; Carmichael, C.; Swift, G.: Single-event upset mitigation for xilinx FPGA block memories. *XILINX Application Note, Virtex-II FPGAs*, 2007.

[12] Nazar, G. L.; Santos, L. P.; Carro, L.: Scrubbing unit repositioning for fast error repair in FPGAs. *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, září 2013, s. 1–10, doi:10.1109/CASES.2013.6662506.

[13] Nunes, J. L.; Cunha, J. C.; Zenha-Rela, M.: Using Failure Prediction to Improve FPGA Scrubbing. *2016 Seventh Latin-American Symposium on Dependable Computing (LADC)*, říjen 2016, s. 135–138, doi:10.1109/LADC.2016.29.

[14] Padovani, R.: Reconfigurable FPGAs in Space – Present and Future. 2005.

[15] Podivinsky, J.; Cekan, O.; Simkova, M.; aj.: The Evaluation Platform for Testing Fault-tolerance Methodologies in Electro-mechanical Applications. *Microprocess. Microsyst.*, ročník 39, č. 8, listopad 2015: s. 1215–1230, ISSN 0141-9331, doi:10.1016/j.micpro.2015.05.011.

[16] Scharoba, S.; Vierhaus, H. T.: An Interactive Design Space Exploration Tool for Dependable Integrated Circuits. *2016 Euromicro Conference on Digital System Design (DSD)*, srpen 2016, s. 714–717, doi:10.1109/DSD.2016.83.

[17] Shastri, A.; Stitt, G.; Riccio, E.: A scheduling and binding heuristic for high-level synthesis of fault-tolerant FPGA applications. *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, červenec 2015, ISSN 1063-6862, s. 202–209, doi:10.1109/ASAP.2015.7245735.

[18] Straka, M.; Kastil, J.; Kotasek, Z.: SEU Simulation Framework for Xilinx FPGA: First Step Towards Testing Fault Tolerant Systems. *14th EUROMICRO Conference on Digital System Design*, IEEE Computer Society, 2011, ISBN 978-0-7695-4494-6, s. 223–230.

[19] Tonfat, J.; Kastensmidt, F. L.; Rech, P.; aj.: Analyzing the Effectiveness of a Frame-Level Redundancy Scrubbing Technique for SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*, ročník 62, č. 6, prosinec 2015: s. 3080–3087, ISSN 0018-9499, doi:10.1109/TNS.2015.2489601.

[20] Vargas, F.; Amory, A.: Transient-fault tolerant VHDL descriptions: a case-study for area overhead analysis. *Proceedings of the Ninth Asian Test Symposium*, prosinec 2000, ISSN 1081-7735, s. 417–422, doi:10.1109/ATS.2000.893659.