SYNCHRONIZACE REKONFIGUROVATELNÉHO SYSTÉMU

Jan Balach
postgraduální studium 1. roč.
Školitel: **prof. Ing. Ondřej Novák, CSc.**

FEL, ČVUT Praha
Technická 2, 166 27 Praha 6 - Dejvice

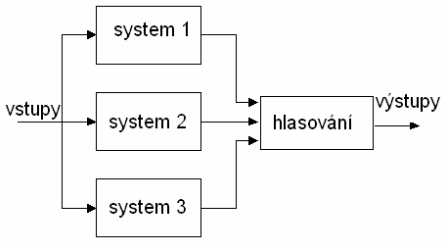
balacj1@fel.cvut.cz

Abstrakt. Tento článek se zabývá analýzou a možnými způsoby synchronizace a obnovy bezporuchových stavů u rekonfigurovatelných fault-tolerant systémů implementovaných v FPGA. Diskutuje možné poruchy, které mohou vzniknout v FPGA, a předkládá možná řešení s analýzou jejich výhod a nevýhod. Dále popisuje experimentální implementaci jednoho z možných řešení, předkládá analýzu daného řešení a diskutuje nad možnostmi jeho dalšího rozšíření.

Klíčová slova. Synchronizace, FPGA, Rekonfigurace.

1 Úvod

V současné době se při návrhu fault-tolerant systémů většinou využívá přidání redundantních prvků, které vykonávají stejnou funkci [1]. Příkladem takové metody návrhu může být například duplexní systém, TMR (Triple Modular Redundancy), jejich modifikace, popř. další. Tyto metody využívají pro tolerování poruch porovnávání výsledků jednotlivých částí. To nám umožní tolerovat poruchy jen do té doby, než převáží počet porouchaných jednotek nad těmi, které ještě pracují bezchybně.



Obrázek 1.1 : schéma TMR

Pokud bychom ale byli schopni opravit porouchanou jednotku a přenést do ní aktuální stav systému, mohli bychom výrazně zlepšit vlastnosti daného systému, jako je např. dostupnost. V článku budou diskutovány druhy poruch, které mohou vzniknout v FPGA (Field Programmable Gate Array) a možnosti, jak dané problémy řešit. Nabídne analýzu výhod a nevýhod představených způsobů a závěrečné shrnutí celé analýzy. Dále představí experimentální návrh implementace jednoho z analyzovaných řešení s diskuzí nad jeho výhodami a nevýhodami.

2 Analýza možností synchronizace a obnovy aktuálního stavu

Synchronizace v našem případě znamená přenos aktuálního stavu systému z jedné jednotky do druhé, opravené, jednotky. Předpokládejme, že systém bude implementován v obvodech FPGA a že máme možnost chybnou jednotku detekovat. U obvodů FPGA může docházet kromě trvalých fyzických defektů (odpojené napájení, mechanické poškození, atd.) také k přechodným poruchám, způsobeným například dopadem nabité částice, tzv. SEU (Single Event Upset). Další přechodné poruchy podobné SEU pak mohou vznikat ojedinělým vlivem okolí, např. krátkodobým naindukováním napětí v důsledku přepětí v síti a podobně. Při opravě systému bychom tedy měli uvažovat, že může nastat libovolná z těchto poruch. Dopady SEU či účinky vnějších vlivů mohou v FPGA vyvolat různé situace. Od náhodných glitchů, které zůstanou nezpozorovány, až po změnu bitů v konfigurační paměti. Vliv SEU na obvody FPGA je diskutován v [2],[3]. V případě vlivu SEU na konfigurační paměť FPGA pak můžeme opravy zasažené jednotky dosáhnout pomocí rekonfigurace. V takovém případě pak bude nutné obnovit informace o aktuálním stavu přenosem z jiné funkční jednotky nebo stabilní zálohy.

2.1 Obnova stavu při poruše mimo konfigurační paměť

Jak bylo již řečeno, přechodné poruchy mohou ovlivnit nejenom konfigurační paměť FPGA, ale také způsobit různé zákmity na vodičích. V takovém případě musíme odlišit mezi dvěma případy. První je ten, kdy je vzniklý impuls tak nepatrný, že není vůbec zaregistrován. V takovém případě se o takové poruše vůbec nedozvíme a tudíž nám nijak nevadí. Situace, kdy nám vzniklý puls na vodičích začne vadit je ten, kdy dojde k jeho zachycení klopnými obvody, a tím i ke změně stavu systému. Je zřejmé, že rekonfigurací zasažené jednotky by došlo k odstranění poruchy. Musíme si ale uvědomit, že časové nároky na rekonfiguraci jsou poměrně vysoké.

Alternativním řešením může být využití obdoby tzv. Micro-rollback architektury, jejíž implementace je popsána v [4]. Tento postup nabízí řešení problému obnovy bezporuchového stavu za cenu mnohem menších časových nároků než rekonfigurace celého FPGA. Předpokládejme, že řízení systému, jehož stav budeme chtít obnovit, je realizováno pomocí stavových automatů. V takovém případě není potřeba funkční kopie zařízení, která poběží s jedno či více taktovým zpožděním, jak je popsáno v referenci. Stav automatu je uchováván v registru. Řešení, které zajistí stejnou funkci, tedy obnovu bezporuchového stavu, tak může zahrnovat pouze jeden záložní registr, který si bude v určitých intervalech vytvářet zálohu bezporuchového stavu. Uvědomme si, že různé aplikace kladou na strukturu automatu rozdílné nároky. Důležitým faktorem je frekvence změny vstupních signálů. Musíme zajistit, aby došlo k obnově aktuálního bezporuchového stavu do doby, než přijde další změna vstupních signálů. Je tedy zřejmé, že čím pomalejší změna vstupních signálu je, tím delší interval mezi vytvářením bodů obnovy je potřeba.

2.2 Obnova stavu při poruše v konfigurační paměti

Dalším projevem přechodné poruchy v FPGA může být změna konfigurační paměti. V takovém případě výše popsaný postup použít nelze, jelikož chybný stav automatů není způsoben vygenerovaným pulsem, ale změnou struktury obvodu v důsledku změny obsahu konfigurační paměti. Při takovém typu poruchy je tedy nutné, aby bylo zasažené FPGA rekonfigurováno. V takovémto případě máme k dispozici několik možností řešení. Můžeme FPGA rekonfigurovat celé a přenést stav z jiného FPGA pomocí synchronizačního protokolu. Jinou možností je modifikace výše popsané metody využívající záložní stav.

2.2.1 Přenesení stavu z bezporuchového FPGA

Jednou z možností, jak lze synchronizovat stav zrekonfigurovaného FPGA s ostatními, je přenos aktuálního stavu z bezporuchového FPGA. Pro toto řešení je nutné navrhnout synchronizační protokol, který zajistí přenesení stavu. Protokol musí využívat asynchronní komunikaci založenou na žádostech a potvrzení, jelikož každé FPGA má vlastní zdroj hodinového signálu. V průběhu synchronizace stavů, musí být obě FPGA necitlivá na vstupní signály. Přenášená data pak mohou být zabezpečena pomocí bezpečnostního kódu (např. Hammingova SEC-DED) nebo pomocí CRC (Cyclic Redundancy Check) či parity. Srovnání některých způsobů je uvedeno v tabulce 2.1, pro výpočet CRC byla zvolena paralelní metoda a implementace převzata z [8]. Zabezpečení zajistí, že v případě poškození přenášených dat bude možné chybu detekovat, popř. opravit. Z tabulky je patrné, že použití zabezpečení přenosu pomocí CRC využívá podstatně více HW prostředků než zabezpečení paritou. Musíme si také uvědomit, že při implementaci rozsáhlejších automatů může rozdíl 15ti LUTů u osmi bitových variant činit kolem 1-2% z celkových HW nároků návrhu. To není příliš velká cena, za poměrně lepší úroveň zabezpečení.

Způsob zabezpečení	Využité FF	Využité LUT
CRC par. po 8bitech	9	19
CRC par. po 32bitech	9	122
CRC par. po 64bitech	9	229
Parita po 8bitech	0	4
Parita po 32bitech	0	11
Parita po 64bitech	0	21

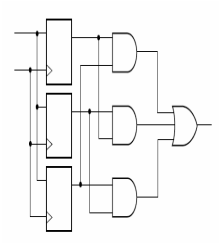
Tabulka 2.1 – Srovnání HW nároků metod pro zabezpečení přenosu

Předpokládáme, že tato metoda je vhodná pro aplikace, kde není změna vstupních signálů příliš četná, tj. vstupní signály se mění v intervalech, které jsou o několik řádů menší, než je frekvence na které běží návrh uvnitř FPGA. To zajistí bezpečné přenesení stavu mezi oběmi FPGA bez rizika ztráty informace. Kritická situace může nastat pouze v případě, že by došlo ke změně vstupních signálů v době, kdy budou přenášena data z jednoho FPGA do druhého. Aby nedošlo ke ztrátě informace, je nutné přidat na vstupní signály buffer, který zachytí případné změny vstupních signálů. Po skončení přenosu se pak provede reakce automatu na posloupnost uloženou v bufferu. Tím se odstraní případná hrozící ztráta informace v průběhu synchronizace. Hlavní nevýhodou se ale jeví hardwarový overhead spojený s obvody zajišťujícími přenos dat a jejich kontrolu.

2.2.2 Využití zálohy bezporuchového stavu

Při tomto řešení předpokládáme modifikaci metody popsané v části 2.1. FPGA se rozdělí na dvě části, statickou a rekonfigurovatelnou. Ve statické části se nachází záložní registr pro uchování bezporuchového stavu. Nekonfigurovatelná část pak obsahuje návrh implementující funkci systému. Abychom zabránili destrukci bezporuchového stavu v důsledku SEU nebo jiného přechodného jevu, je nutné implementovat registr pro zálohu stavu metodou, která mu zajistí vetší odolnost proti takovým vlivům. Metoda implementace registru odolného vůči dopadu SEU je popsána v [5]. Další možností, jak zabezpečit registr, je využití bezpečnostních kódů. To však zvýší hardwarový overhead návrhu hlavně co se kombinační logiky týče. S tím je samozřejmě spojená i maximální frekvence, na které

může obvod fungovat. Je tedy otázka, co je pro nás výhodnější. Můžeme rovněž využít kombinace jednoduchého kódu (např. parity) se zabezpečením registru znázorněném na obrázku 2.1.

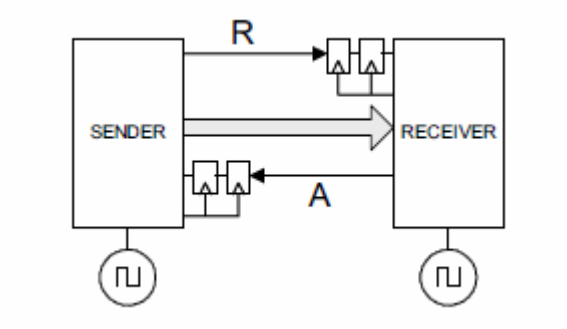


Obrázek 2.1: Schéma zabezpečení registru proti SEU

V případě zásahu konfigurační paměti SEU by tedy došlo k rekonfiguraci návrhu a následnému přenesení stavu ze záložního registru. Problém tohoto řešení je možné rozsynchronizování opravovaného FPGA s ostatními. Důvodem je doba potřebná pro rekonfiguraci, která je řádově několik milisekund. U některých aplikací se za tu dobu mohou změnit vstupní signály a po provedení opravy by pak FPGA udávalo jinou informaci než ostatní, tudíž by bylo opět označené za porouchané. Daný problém se dá odstranit použitím bufferu na vstupní signály. Tyto buffery musejí být uloženy ve statické části FPGA a rovněž zabezpečeny proti efektům SEU, např. zdvojením s kontrolou výstupů hradlem XOR. Tato metoda je rovněž popsána v [5]. Využitím bufferů se zajistí, aby nedošlo ke ztrátě informace v průběhu rekonfigurace. Výhoda tohoto systému je relativně malý hardwarový overhead.

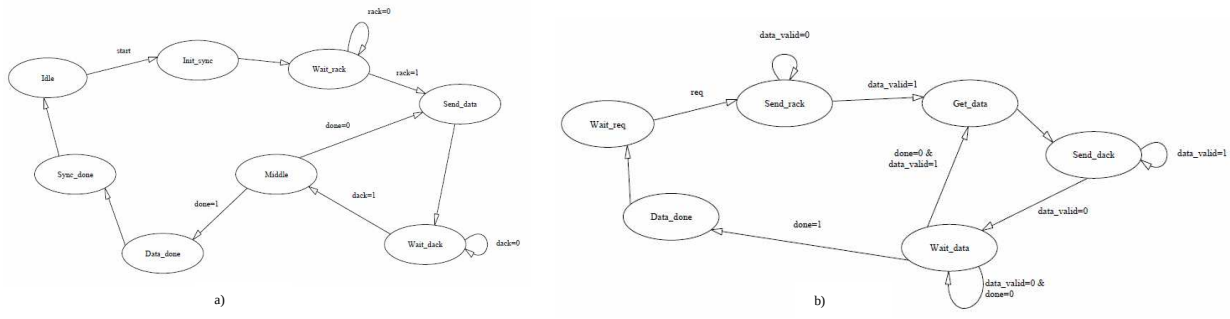
3 Návrh experimentálního obvodu pro synchronizaci

Na základě analýzy možných řešení byl vytvořen expermintální obvod pro synchronizaci stavu mezi dvěma FPGA. Obvod je implementován metodou přenosu stavu z funkčního FPGA, která je popsána v sekci 2.2.1. Přenos dat mezi FPGA je realizován serio-paralelně. Data pro přenos jsou rozdělena do Bytů, které jsou pak přenášeny z jednoho FPGA do druhého. Propojení FPGA je implementováno pomocí modelu popsaného v [7]. Tím je dosaženo odstranění metastability na řídích signálech. Schéma propojení je zobrazeno na obrázku 3.1



Obrázek 3.1: Propojení FPGA

Při implementaci byl jako první vytvořen funkční model jednoduchého protokolu pro synchronizaci. Protokol implemetuje komunikaci mezi dvěma automaty, příjemcem a odesílatelem. Komunikace mezi nimi je realizována na základě žádostí a potvrzování. Žádost o synchronizaci zahajuje vždy příjemce. Grafy přechodů obou automatů jsou vidět na obrázku 3.2



Obrázek 3.2: Grafy přechodu a) odesílatele b) příjemce

Hardware zajišťující synchronizaci podle navrženého protokolu pak obsahuje mimo těchto dvou automatů a potřebných obvodů pro přepínání vstupů stavového registru systému ještě třetí, velmi jednoduchý automat, který celý systém řídí. Automat má čtyři hlavní stavy, *Functional*, *Reseted*, *Sending* a *Recieving* a několik pomocných, které řeší krajní podmínky, které mohou nastat, jako je např. resetování všech zařízení najednou.

Přenášená data jsou zabezpečena pomocí 8-bitového CRC, které se počítá paralelně po 8-bitech. Vypočtené CRC se připojí jako poslední Byte přenášených dat, pro jeho výpočet byl zvolen polynom $x^8 + x^2 + x + 1$. Pokud není počet přenášených bitů dělitelný 8 beze zbytku, jsou chybějící bity doplněny nulami. Příjemce v průběhu ukládání dat provádí kontrolu CRC. Princip kontroly je prakticky stejný jako samotný výpočet. Přijaté Byty jsou zpracovány stejným obvodem, který odesílatel používá pro výpočet. Nakonec obvod dostane na vstup i zaslané CRC a dokončí kontrolní výpočet. Pokud jsou přenesená data v pořádku, výsledek výpočtu jsou samé nuly. Dojde-li k chybě, žádá centrální automat na straně příjemce o opakování přenosu synchronizačních dat. Použitý obvod pro výpočet CRC byl převzat z [8].

Pro implementaci bylo vybrán jednoduchý automat, který implementuje jeden z univerzálních bloků pro stavbu železniční stanice. Tyto bloky přepsal z reléové realizace do VHDL Martin Zatřepálek ve své diplomové práci [9]. V tabulce 3.1 můžeme vidět výsledky syntézy provedené programem Xilinx ISE 10.1.

Blok	Využití FF	Využití LUT	Hodinová perioda [ns]
K + sync. HW	60	298	6,339
K bez sync. HW	18	94	4,600

Tabulka 3.1 – Využití prostředků FPGA

3.1 Testování funkce obvodu

Celý obvod byl testován na několika úrovních. Nejprve došlo k otestování komunikačního protokolu na úrovní abstraktního modelu. K tomuto účelu byl zvolen model Petriho sítě [6]. Petriho síť je silný abstraktní nástroj pro modelování systému. Pomocí Petriho sítě byl odsimulován komunikační protokol a bylo ověřeno, že v něm nemůže dojít uváznutí. Následně byl celý systém implementován a funkce otestována v programu Modelsim. Celý návrh byl testován po částech i jako celek, aby byla prověřena jeho správná funkce.

3.2 Analýza systému

Testy prokázaly funkci systému na úrovní behaviorální simulace. Při pohledu do tabulky 3.1 můžeme vidět výsledky implementace experimentálního systému. Je patrné, že HW overhead je velice vysoký, přibližně 200%. To je způsobeno tím, že automat realizující blok K, je poměrně malý. Dá se předpokládat, že s rostoucím automatem, jehož stav budeme chtít obnovovat, se bude relativní overhead snižovat, protože jediné, co se bude měnit bude šířka sběrnic a některých multiplexorů. HW overhead by se dal také ovlivnit kódováním automatů zajišťujících přenos dat. To je zatím 1 z N, možnou změnou kódování by se dalo uspořit několik LUTů. Toto je potřeba ještě experimentálně ověřit. Při pohledu na časový overhead můžeme vidět, že i přes vysoký HW overhead, je poměrně malý, asi 37%. To je s největší pravděpodobností způsobeno přídavnými multiplexory do kritické cesty automatu. Předpokládáme, že při implementaci větších automatů se tento overhead už moc měnit nebude. Je ale třeba to ještě experimentálně ověřit.

Celková doba potřebná pro synchronizaci se dá rozdělit do tří částí - navázání komunikace, přenos dat a ukončení komunikace. Doba navázání komunikace je vždy konstantní a trvá devět taktů, pak je zahájeno odesílání prvního Bytu. Doba potřebná na přenesení jednoho Bytu je šest taktů. Ukončení přenosu pak zabere tři takty. Z této analýzy můžeme vytvořit vztah $T_{prenosu} = 12 + 6 * pocet\_Byte$ *taktů* . Ve výpočtu není zahrnut čas potřebný na výpočet CRC u přenesených dat. Důvodem je způsob výpočtu CRC, který probíhá paralelně s přípravou a přenosem dat. Po obdržení žádosti o zaslání synchronizačních dat začne odesílatel připravovat data do výstupního bufferu a zároveň počítat CRC. Proto je již před odesláním posledního Byte kontrolní výpočet dokončen.

Z uvedeného vztahu je patrné, že časová náročnost potřebná pro synchronizaci tímto způsobem lineárně závisí na velikosti dat, které musíme přenést. Je tedy vidět, že celý přenos dat většího systému trvá poměrně dlouhou dobu. Proto se nabízí možnost využít kombinace obou metod pro obnovu stavu, a to zálohy bezporuchového stavu pomocí registru v kombinaci s přenosem dat z druhého FPGA.

Při využití kombinace obou metod musíme vzít do úvahy, že zabezpečení statické části bude odolné vůči efektům SEU pouze určitou dobu. Proto je nutné po několika rekonfiguracích a obnovách stavu pomocí záložního registru využít možnost rekonfigurace celého FPGA a obnovy stavu zajistit přenosem z bezporuchového FPGA. Tím dojde k odstranění případných, již tolerovaných, poruch ve statické části FPGA. Tento způsob ušetří čas zejména u větších obvodů, kde jsou časové nároky přenosu stavů mezi FPGA poměrně veliké. Proti tomuto řešení mluví hlavně potřebný hardwarový overhead. Můžeme předpokládat, že bude poměrně vysoký vzhledem k nutnosti implementovat kombinaci obou způsobů. Tyto hypotézy je nutno ještě experimentálně ověřit.

4 Téma disertace

V budoucím studiu bych se chtěl zaměřit na návrh obvodů se zvýšenou spolehlivostí. Hlavně se zaměřením na hradlová pole FPGA a proces jejich rekonfigurace. V současné době se rekonfigurace za účelem zvýšení spolehlivosti provádí jen výjimečně a v nejjednodušších případech, předpokládaný výzkum by měl rozšířit možnosti použití na komplexnější úlohy. Daným tématem jsem se již z části zabýval ve své diplomové práci [10], kde jsem se zkoumal využití rekonfigurace za účelem testování více-jádrové aplikace.

5 Shrnutí

Byly zde prezentovány myšlenky na možné zlepšení vlastností fault-tolerant systémů založených na využití redundantních částí. Uvedené způsoby se zabývají obnovou bezporuchového stavu u opravené

jednotky systému. Jednotlivé způsoby jsou podrobně analyzovány s možnou diskuzí o způsobu jejich implementace. Dále jsou rozebírány a diskutovány výhody a nevýhody daných řešení. U diskutovaných nevýhod jsou překládána možná řešení pro jejich odstranění nebo na jejich kompenzaci.

Dále byl představen experimentální návrh, který implementuje jeden z prezentovaných způsobů pro obnovu stavu. Daný návrh má poměrně vysoký HW overhead, ale časový overhead je vzhledem k HW poměrně malý. Výsledky implementace a důvody, proč je HW overhead tak vysoký byly analyzovány současně s časovou analýzou přenosu dat. Dále je diskutováno možného zlepšení dosažených výsledků společně s dalšími možnými rozšířeními systému.

Literatura

[1] D.K.Pradhan: Fault-Tolerant Computer System Design, Prentice Hall, 1996

[2] QuickLogic Corporation: Single_Event_Upsets_in_FPGAs.pdf, 2003, www.quicklogic.com

[3] Xilinx: xilinx_radiation_effects.pdf , www.xilinx.com

[4] M. Pflanz, F. Pompsch, H.T. Vierhaus: An efficient On-line-Test and Back-up Scheme for embedded Processors, Proc. Intl. Test Conference, 1999, Atlantic City, pp. 964-972

[5] Xilinx: XAPP181.pdf, SEU Mitigation Design Techniques for the XQR4000XL, www.xilinx.com

[6] Z. Hanzálek: Petriho sítě, ČVUT v Praze, červen 2008

[7] Ran Ginosar: Fourteen Ways to Fool Your Synchronizer, Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC'03) 1522-8681/03

[8] Geir Drange: Ultimate CRC, 2005, www.opencores.org

[9] Zatřepálek Martin: Zabezpečovací zařízení pro železniční stanici založené na FPGA, diplomová práce, 2008, ČVUT v Praze

[10] Balach Jan: Diagnostický systém pro obvod realizovaný na FPGA, diplomová práce, 2008, ČVUT v Praze