VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

METODIKA NÁVRHU SYNCHRONIZACE A OBNOVY STAVU SYSTÉMU ODOLNÉHO PROTI PORUCHÁM

METHODOLOGY FOR FAULT TOLERANT SYSTEM STATE SYNCHRONIZATION DESIGN AND ITS RECOVERY FROM FAULTS

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE — **Ing. KAREL SZURMAN**
AUTHOR

ŠKOLITEL — **Doc. Ing. ZDENĚK KOTÁSEK, CSc.**
SUPERVISOR

BRNO 2020

Abstrakt

Tato disertační práce představuje metodiku vytvořenou pro návrh synchronizace a obnovy stavu systému odolného proti poruchám. Metoda synchronizace stavu navržená podle popsané metodiky umožňuje opravit stav paměťových prvků systému, které jsou implementovány v aplikační logické vrstvě číslicového návrhu v FPGA a jejichž hodnoty nelze opravit částečnou dynamickou rekonfigurací. Vytvořená metodika popisuje možné způsoby návrhu metod synchronizace s ohledem na granularitu TMR, závislost funkce systému na předchozích stavech a samotné architektuře číslicového systému. Metodika se blíže zaměřuje na hrubozrnné architektury TMR a problematiku synchronizace stavu v systémech řízených stavovými automaty nebo procesorem. V této práci je využití vytvořené metodiky předvedeno na návrhu metod synchronizace stavu pro systém řadiče sběrnice CAN odolného proti poruchám a zabezpečený systém mikrokontroléru NEO430. Při experimentálním ověření mechanismů opravy a obnovy stavu systému po poruše byla ověřena jak správná funkce systémů, tak jejich spolehlivost v přítomnosti simulovaných poruch typu SEU. V závěru práce jsou diskutovány dosažené experimentální výsledky a přínos práce.

Abstract

In this Ph.D. thesis, a new methodology for the fault tolerant system state synchronization design and its recovery from faults is presented. A state synchronization method designed by means of the proposed methodology allows to repair the state of sequential logic elements implemented in the FPGA application logic, which cannot be repaired by the partial dynamic reconfiguration. The proposed methodology describes possible state synchronization design methods with respect to TMR granularity, dependence of the system function on its previous states and the system architecture. The methodology focuses on coarse-grained TMR architectures and state synchronization in the systems controlled by means of finite state machines or a processor. The use of the methodology is demonstrated on the CAN bus control system and the microcontroller NEO430, for which specific synchronization methods were designed. The systems reliability and new ability of the systems for recovery from faults were verified in the presence of simulated SEU faults. The experimental results and the contribution of this thesis are discussed in the conclusion.

Klíčová slova

synchronizace stavu, obnova stavu, částečná rekonfigurace, systém odolný proti poruchám, spolehlivost, dostupnost, SEU, TMR, FPGA, sběrnice CAN, mikrokontrolér NEO430

Keywords

state synchronization, state recovery, partial reconfiguration, fault tolerant system, dependability, availability, SEU, TMR, FPGA, CAN bus, microcontroller NEO430

Citace

SZURMAN, Karel. *Metodika návrhu synchronizace a obnovy stavu systému odolného proti poruchám.* Brno, 2020. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Školitel Doc. Ing. Zdeněk Kotásek, CSc.

Metodika návrhu synchronizace a obnovy stavu systému odolného proti poruchám

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením pana Doc. Ing. Zdeňka Kotáska, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Karel Szurman
31. srpna 2020

Poděkování

Děkuji mému školiteli, panu Doc. Ing. Zdeňkovi Kotáskovi, CSc. za odborné vedení, cenné konzultace a rady při tvorbě této disertační práce a že mi byl oporou během celé doby studia. Děkuji své manželce Monice, za pochopení a nekonečnou podporu. Dále děkuji mým kolegům z UPSY, kteří se mnou spolupracovali.

Obsah

1 Úvod **4**
- 1.1 Předmluva . . . 4
- 1.2 Motivace . . . 6
- 1.3 Organizace práce . . . 7

2 Principy systémů odolných proti poruchám **8**
- 2.1 Spolehlivost systémů . . . 8
  - 2.1.1 Hrozby pro spolehlivost systému . . . 10
  - 2.1.2 Ukazatelé spolehlivosti . . . 10
- 2.2 Poruchy číslicových systémů a obvodů FPGA . . . 13
  - 2.2.1 Radiační prostředí okolo Země . . . 14
  - 2.2.2 Vliv kosmického záření na polovodičové součástky . . . 14
  - 2.2.3 Jednorázové poruchy Single Event Effects . . . 15
- 2.3 Rekonfigurovatelné obvody FPGA . . . 17
  - 2.3.1 Technologie obvodů SRAM FPGA . . . 17
  - 2.3.2 Architektura obvodu FPGA Xilinx Virtex-5 . . . 20
  - 2.3.3 Implementace návrhu číslicového systému do FPGA . . . 23
  - 2.3.4 Částečná dynamická rekonfigurace . . . 25
  - 2.3.5 Model poruch obvodu SRAM FPGA . . . 27
- 2.4 Koncepce systémů odolných proti poruchám . . . 29
- 2.5 Využití obvodové redundance pro detekci a maskování poruch . . . 31
  - 2.5.1 Duplexní systém s porovnáním . . . 31
  - 2.5.2 Systém TMR . . . 33
- 2.6 Oprava poruch v obvodech SRAM FPGA pomocí rekonfigurace . . . 33
  - 2.6.1 Periodické ověřování a čištění konfigurační paměti . . . 35
  - 2.6.2 Oprava systému z permanentní poruchy . . . 38
- 2.7 Zotavení stavu systému z poruchy . . . 39
- 2.8 Formální definice problému synchronizace stavu . . . 40

3 Současný stav řešeného problému **42**
- 3.1 Synchronizace TMR na úrovni redundantních obvodů FPGA . . . 42
- 3.2 Synchronizace na úrovni víceprocesorového systému . . . 43
- 3.3 Synchronizace TMR na úrovni procesoru . . . 45
- 3.4 Synchronizace TMR na úrovni konečných automatů . . . 48
- 3.5 Synchronizace TMR na úrovni registrů sekvenční logiky . . . 49
- 3.6 Shrnutí . . . 52

4 Cíle disertační práce **53**

4.1 Návaznost disertační práce na již existující metodiky 53
4.2 Stanovení cílů 53

5 Návrh opravovaného systému odolného proti poruchám 55
5.1 Zabezpečení systému pomocí TMR 55
5.1.1 Pokročilé metody implementace TMR 55
5.1.2 Principy implementace a vkládání majoritního voliče 58
5.1.3 Automatické generování TMR 61
5.2 Principy návrhu rekonfigurovatelného systému v FPGA 64
5.2.1 Rekonfigurovatelný návrh systému 64
5.2.2 Návrhový proces pro obvody FPGA od firmy Xilinx 65
5.3 Oprava stavu systému pomocí rekonfigurace FPGA 69
5.3.1 Řadič rekonfigurace GPDRC 70
5.4 Simulace poruch v konfigurační paměti FPGA pomocí injektoru poruch typu SEU 72

6 Metodika synchronizace stavu rekonfigurovatelných modulů 74
6.1 Návrh systému s využitím metodiky synchronizace stavu 74
6.1.1 Aplikace metodiky během návrhového procesu systému 74
6.1.2 Výběr metody synchronizace 76
6.2 Principy návrhu metod synchronizace stavu 77
6.2.1 Krok 1: Výběr vhodného synchronizačního stavu 78
6.2.2 Krok 2: Definice synchronizovaných objektů stavu systému 79
6.2.3 Krok 3: Návrh propojení hardwarových jednotek 80
6.3 Základní architektura synchronizačních obvodů 81
6.4 Parametry metod synchronizace stavu 82
6.4.1 Dynamické parametry 82
6.4.2 Statické parametry 82
6.5 Proces opravy stavu systému pomocí rekonfigurace a synchronizace 83
6.6 Shrnutí 84
6.6.1 Předpoklady pro implementaci metody synchronizace dle navržené metodiky 84

7 Synchronizace stavu systému s dávkovým zpracováním dat 85
7.1 Návrh řadiče sběrnice CAN odolného proti poruchám 86
7.1.1 Řadič sběrnice CAN 86
7.1.2 Rekonfigurovatelná architektura odolná proti poruchám 88
7.2 Návrh techniky synchronizace stavu řadiče s využitím navržené metodiky . 89
7.2.1 Implementace logických obvodů pro synchronizaci stavu 90
7.2.2 Synchronizace stavu komunikační vrstvy řadiče 90
7.2.3 Synchronizace stavu aplikační vrstvy řadiče 90
7.3 Experimentální a implementační výsledky 92
7.4 Shrnutí 93

8 Synchronizace operačního stavu procesorového systému 95
8.1 Synchronizace stavu pro systém řízený procesorem 95
8.1.1 Stav procesoru 96
8.1.2 Synchronizace během zpracování instrukcí procesoru 96
8.1.3 Návrh metod synchronizace stavu procesoru 97

8.1.4 Programová synchronizace stavu procesorů . . . 100
8.2 Návrh a implementace procesorového systému odolného proti poruchám . . 102
8.2.1 Mikrokontroler NEO430 . . . 102
8.3 Návrh zabezpečené architektury ReSyTMR . . . 103
8.3.1 Rekonfigurovatelná architektura . . . 107
8.4 Implementace metod pro synchronizaci . . . 108
8.4.1 Synchronizace pomocí synchronizačního restartu . . . 108
8.4.2 Synchronizace pomocí sdílené datové paměti . . . 109
8.5 Experimentální a implementační výsledky . . . 113
8.5.1 Testovací prostředí . . . 113
8.5.2 Nároky na implementaci architektury ReSyTMR . . . 115
8.5.3 Vyhodnocení odolnosti proti poruchám . . . 116
8.6 Shrnutí . . . 119

9 Závěr **120**
9.1 Shrnutí výsledků práce . . . 120
9.2 Přínos práce . . . 121
9.3 Možné rozšíření práce . . . 123
9.3.1 Synchronizace stavu při opravě trvalé poruchy pomocí relokace . . . 123
9.3.2 Synchronizace stavu pomocí vyčtení stavu registrů z FPGA . . . 123

Literatura **125**

Kapitola 1

Úvod

Spolehlivost je jedním z nejdůležitějších parametrů dnešních elektronických systémů, od jednoduchých integrovaných obvodů po komplexní řídicí systémy. Spolehlivost systémů je převážně ovlivněna působením vnějšího prostředí nebo stárnutím a opotřebením elektronických součástek. Správný a metodický návrh by měl zajistit bezpečný provoz a správnou funkci systému po celou dobu jeho životnosti. Nicméně pro bezpečnostně kritické systémy, které vykonávají velmi důležitou funkci, je nutné zajistit jejich provozuschopnost i v případě přítomnosti poruch. Příkladem mohou být letecké řídicí systémy vykonávající funkci kritickou pro samotný let letadla nebo vesmírné družice vyslané na oběžnou dráhu kolem Země či do hlubokého vesmíru. Takové systémy jsou tedy konstruovány jako odolné proti poruchám [84]. Kosmické záření je jedním z hlavních jevů, který může ohrozit funkci těchto elektronických zařízení [11]. Kosmické záření je tvořeno vysokoenergetickým proudem částic, který do zemského povrchu proniká z kosmického prostoru. Různě nabité částice vytváří ionizované radiační prostředí, které působí na fyzikální a chemické vlastnosti látek použitých v polovodičových součástkách.

1.1 Předmluva

Technologický pokrok v oblasti vývoje křemíkových čipů dnes dosahuje limitů integrace, které byly dříve nepředstavitelné. Počet tranzistorů integrovaných na jednom čipu exponenciálně vzrostl a složitost číslicových obvodů se mnohonásobně zvětšila.

Gordon Moore v roce 1965 vyslovil myšlenku, podle které se maximální počet tranzistorů umístěných na jednom čipu zdvojnásobí každý rok. V roce 1975 upřesnil svou předpověď, že se tomu tak stane každé dva roky[1] [113]. Vývoj číslicových obvodů, programovatelných logických obvodů a zejména mikroprocesorů pokračoval ve smyslu této myšlenky, tzv. Moorova zákona. Díky zmenšení velikosti tranzistorů bylo v dalších letech snahou navyšování počtu komponent umístěných na jednom čipu. Technologický pokrok umožnil zvýšit hustotu tranzistorů a navrhnout čipy s vysokým stupněm integrace (LSI) a velmi vysokým stupněm integrace (VLSI) polovodičových prvků na ploše čipu.

V roce 1971 byl vyvinut firmou Intel první komerční mikroprocesor Intel 4004 [29]. Mikroprocesor obsahoval procesorovou jednotku CPU složenou z cca. 2300 tranzistorů a doplňující integrované obvody implementující paměť ROM, operační paměť RAM, posuvné registry a vstupně výstupní porty.

[1] Moorův odhad byl následně upřesněn na 18 měsíců při uvážení zvyšující se rychlosti tranzistorů.

Jako alternativa k mikroprocesorům, které na fixních prvcích umožňují provádět uživatelský program, vznikly logické obvody programovatelné na obvodové úrovni. První programovatelné obvody PLD (Programmable Logic Device) se objevily v polovině 70. let. Během následujících let vznikly různé typy programovatelných obvodů. Obvody FPLA (Field Programmable Logic Arrays) obsahovaly dvě programovatelné matice hradel AND a hradel OR. Takové zapojení bylo vzhledem k zapojení hradel v kaskádě pomalé. Jako odpověď vznikly obvody PAL (Programmable Array Logic), které byly složeny z programovatelné matice hradel AND a fixní matice hradel OR. Tyto obvody využívaly jednorázově programovatelné paměti PROM (Programmable Read Only Memory) [3].

S vývojem nových typů přeprogramovatelných pamětí EPROM (Erasable PROM) a EEPROM (Electrically Erasable PROM), vznikly v 80. letech programovatelné obvody GAL (Generic Array Logic) umožňující několikanásobné přeprogramování propojení logických hradel. Obvody GAL měly speciální výstupní obvod s dodatečným klopným obvodem, log. hradlem XOR a několika multiplexory, umožňující rozšíření naprogramované funkcionality [73]. S potřebou na realizaci složitějších logických funkcí vznikly obvody CPLD (Complex PLD), které kombinovaly několik obvodů PAL na jednom čipu. Systémy vytvořené v praxi byly implementované za pomocí kombinační logiky programovatelných obvodů a sekvenční logiky tvořené různými typy klopných obvodů a pamětí [3].

Počátkem 80. let se na trhu objevily také zákaznické integrované obvody ASIC (Application Specific Integrated Circuit). Na rozdíl od mikroprocesorů a programovatelných obvodů vhodných pro široké použití, mohou být obvody ASIC navrženy a optimalizovány přesně na míru dané aplikaci [28]. Obvody ASIC se zapříčinily o rychlý rozmach polovodičového průmyslu a později hrály důležitou roli v rozvoji telekomunikačních systémů [97]. Omezením obvodů ASIC je jejich drahá výrobní technologie, díky čemuž se vyplatí pouze při velkých výrobních sériích.

V roce 1985 vytvořila firma Xilinx první obvod FPGA XC2064 s výrobní technologií 2.5μm. FPGA XC2064 obsahovalo 64 konfigurovatelných logických bloků (CLB) složených ze dvou tří-vstupových vyhledávacích tabulek (LUT) a jednoho registru [97]. První obvody FPGA implementovaly víceúrovňovou architekturu, která rozdělovala logické bloky FPGA do vzájemně oddělených skupin. Nedostatkem těchto obvodů byl neefektivní návrh propojení a využití logických bloků, což ztěžovalo jejich používání. Obvody FPGA bylo z počátku díky jejich malé velikosti možné programovat ručně. Následně byla architektura obvodů FPGA zdokonalena do dnešní podoby, vytvořením 2D matice s rovnoměrným uspořádáním logických bloků [3]. Taktéž se objevili jednou programovatelné obvody FPGA složené z propojek (anti-fuse). Na FPGA s technologií Antifuse se specializovala firma Actel (nyní Microsemi). Tyto obvody byly rychlé a po naprogramování měly pevnou konfiguraci. První obvod ACT1 se objevil v roce 1989 [20]. V 90. letech trh s obvody FPGA rychle rostl. Firma Xilinx začala využívat ve svých obvodech FPGA volatilní konfigurační paměť SRAM. Tato technologie umožňovala neomezené přeprogramování FPGA, ale s nutností uložení konfigurace v dodatečné paměti. Firma GateField (později Actel, a nyní Microsemi) nabídla alternativní technologii obvodů FPGA s nevolatilní a přeprogramovatelnou konfigurační pamětí Flash [3].

V polovině 90. let se průmysl výroby křemíkových čipu začal soustředit na škálování výkonnosti zvyšováním maximální hodinové frekvence. Vedlejším efektem zvyšování frekvence bylo daleko vyšší vyzařování tepla na úrovni toho, co digitální čipy mohly snést. Vývoj mikroprocesorů se následně soustředil na zvyšování počtu procesorových jader.

Trendem následujících let v oblasti vývoje digitálních obvodů byl růst počtu programovatelných hradel, zvýšení rychlosti, integrace speciálních funkčních bloků (např. DSP,

násobičky), využití pamětí SRAM a Flash. V oblasti vývoje složitějších číslicových obvodů a zákaznických čipů se objevily také první systémy na čipu (SoC). Se zvyšující se složitostí digitálních obvodů byly navrženy nové způsoby popisu obvodů na vyšších úrovních abstrakce. Na konci 90. let bylo použití nástrojů pro automatickou syntézu, rozmístění a propojení vzhledem ke komplexnosti a velikosti návrhu logiky obvodů umístěných do FPGA nutností [97]. Rovněž vznikly nové návrhové techniky pro snadnou testovatelnost a diagnostiku číslicových obvodů.

Dnešní elektronické systémy obsahují množství digitálních obvodů a často kombinují mikroprocesory, zákaznické obvody ASIC a programovatelné obvody FPGA. Vývoj mikroprocesorů se od svého počátku rozdělil na několik oblastí: procesory pro obecné výpočty (současná 10nm výrobní technologie procesorů umožňuje integrovat 100 milionů tranzistorů na cm^2 [19]), grafické procesory (GPU), mikrokontrolery určené pro vestavěné systémy (integrující mikroprocesor a periferní zařízení na jednom čipu), digitální signálové procesory (DSP) a aplikačně specifické procesory (ASIP). Obvody FPGA si našly své uplatnění hlavně díky nižší ceně, vysoké flexibilitě umožňující jednodušší odladění číslicového návrhu, snadnou verifikaci a rychlou adaptaci návrhu na změny ve specifikaci systému [97] [31]. Dnes se oblast použití obvodů ASIC a FPGA často překrývá. Obvody FPGA mohou být použity jako platforma pro ověření návrhu předtím, než se vyrobí ASIC. Firma Xilinx představila v roce 2019 nové obvody FPGA zvané ACAP (Adaptive Compute Acceleration Platform) vyrobené za pomocí 7nm technologie. Cílem obvodů ACAP je poskytnout uživateli univerzální platformu umožňující skloubit funkcionalitu procesorů pro obecné výpočty, vektorové zpracování jako u DSP nebo GPU procesorů a programovatelnou logiku FPGA v rámci jedné architektury [109].

1.2 Motivace

Nicméně, s novými technologiemi přišly i nové výzvy a problémy, kterým museli návrháři číslicových obvodů čelit. Zmenšování tranzistorů vedlo k omezování pracovního napětí, zvyšování frekvencí a ovlivnění spolehlivosti integrovaných obvodů. Spolehlivost se stala jedním z klíčových parametrů moderních systémů. Perfektní systém se 100% spolehlivostí je prakticky nemožné vytvořit. Důvodem je, že s rostoucí složitostí systému výrazně klesá jeho spolehlivost [23]. Tímto problémem se již v 50. letech začal zaobírat John von Neumann, který následně publikoval výzkumný článek na téma vytvoření spolehlivého systému z nespolehlivých komponent a představil myšlenku využití redundance ke zvýšení spolehlivosti systému [69]. Redundance, implementována v různých podobách, se stala základním nástrojem systémů odolných proti poruchám. V roce 1975 byl zdokumentován první případ poruchy vesmírného satelitu způsobené kosmickým zářením [70]. V roce 1978 byl výzkumníky z firmy Intel publikován článek popisující nový typ poruch, vzniklých vlivem kosmického záření, způsobující tzv. měkké chyby (*soft-errors*) v pamětech typu DRAM [62]. Tato porucha byla definována jako náhodná, jednorázová porucha jednoho bitu v paměti nezpůsobená elektrickým nebo elektromagnetickým rušením, ale zářením. Dnes jsou na měkké chyby nejvíce citlivé paměti SRAM.

Obvody SRAM FPGA se postupně staly oblíbenou obvodovou platformou pro různorodé aplikace ve všech průmyslových odvětví trhu. Výhodou moderních obvodů SRAM FPGA je možnost částečné dynamické rekonfigurace PDR (Partial Dynamic Reconfiguration), která umožňuje návrhářům změnit část implementované aplikace v FPGA a přitom ponechat celý systém v provozu. Díky PDR lze implementovat systém, který změní nebo přizpůsobí svou funkcionalitu vnějším podmínkám. Dalším příkladem může být minimalizace spotře-

bovaných prostředků v FPGA, kdy se pomocí časového multiplexu přeprogramuje funkce dílčího subsystému. PDR lze také použít v systémech odolných proti poruchám pro opravu poruch v konfigurační paměti FPGA. Obvody FPGA se používají také v leteckých a vesmírných systémech, a taktéž v ostatních systémech s vysokou spolehlivostí [49]. Návrhové a vývojové postupy pro bezpečnostně kritické systémy vyžadují použití technik pro zvýšení spolehlivosti a zmírnění následků poruch způsobených kosmickým zářením.

Systémy odolné proti poruchám jsou často konstruovány jako tzv. duplexní systémy nebo systémy TMR. Znamená to, že vlastní aplikace je zdvojena či je dokonce realizována třikrát. Je pak zajištěno, že výstupy jednotlivých implementací redundantních modulů jsou srovnávány, na výstup se pak dostává pouze správný výsledek. Pokud je systém odolný proti poruchám implementován do FPGA, pak ta část, v níž byla rozpoznána porucha, bude následně rekonfigurována. Po dobu rekonfigurace bude nefunkční a po skončení rekonfigurace musí být uvedena do stavu, který bude v souladu se stavem zbývajících implementací (které byly po celou dobu rekonfigurace funkční a plnily svou roli). Dá se tudíž hovořit o synchronizaci stavu hardwarových jednotek po vzniku poruchy. Zatímco rekonfigurace umožňuje opravu přechodných i trvalých poruch v konfigurační paměti FPGA, synchronizace stavu umožňuje opravit i stav paměťových elementů v sekvenčních obvodech systému. Cílem této disertační práce je vytvoření metodiky pro návrh metod synchronizace stavu číslicových systémů odolných proti poruchám, které umožní zvýšení jejich spolehlivosti a dostupnosti v přítomnosti poruch.

1.3 Organizace práce

Tato disertační práce je organizována následovně. Principy návrhu systémů odolných proti poruchám jsou popsány v kapitole 2. Tato kapitola se zaměřuje na problematiku implementace systémů odolných proti poruchám do obvodů FPGA s konfigurační pamětí SRAM. Tyto obvody jsou citlivé na vznik dočasných i trvalých poruch způsobených vlivem kosmického záření. V kapitole je popsán model uvažovaných poruch a způsoby zabezpečení číslicových systémů, které umožňují poruchy tolerovat i opravit. V kapitole 3 jsou popsány existující techniky pro synchronizaci stavu systému. Existující metody pro synchronizaci stavu jsou rozděleny do několika kategorií podle úrovně abstrakce návrhu, na které je implementována architektura TMR a provedena synchronizace mezi redundantními moduly.

Cíle této disertační práce jsou definovány v kapotole 4. Kapitola 5 popisuje způsob řešení návrhu opravovaného systému odolného proti poruchám s využitím implementace architektury TMR a schopnosti částečné dynamické rekonfigurace obvodů FPGA. V této kapitole jsou také blíže popsány principy implementace hrubozrnné a jemnozrnné architektury TMR. V neposlední řadě je zde popsán využitý řadič rekonfigurace, nástroj BYU BL-TMR [18] pro automatické generování jemnozrnné architektury TMR a nástroj pro injekci poruch typu SEU do konfigurační paměti. Kapitola 6 obsahuje popis metodiky pro návrh synchronizace stavu rekonfigurovatelných modulů. Kapitoly 7 a 8 se zaměřují na aplikaci vytvořené metodiky pro návrh synchronizace v systémech řízených stavovým automatem a systémech řízených procesorem. Metodika je v těchto kapitolách rozvedena vzhledem ke specifickým vlastnostem a rysům architektur těchto systémů. Následně jsou popsány implementované metody pro synchronizaci stavu, experimenty ověřující správnou funkci implementovaných opravných mechanismů a spolehlivostní parametry ověřené pomocí injekce poruch typu SEU. Dosažené výsledky experimentů jsou vzájemně porovnány.

Kapitola 9 shrnuje celý výzkum a přínosy této disertační práce. Dále se kapitola zabývá dalším pokračováním výzkumu a rozšířením práce.

Kapitola 2

Principy systémů odolných proti poruchám

Tato kapitola představuje čtenáři problematiku návrhu systémů odolných proti poruchám a jejich implementaci do obvodů FPGA. Kapitola obsahuje základní rámec znalostí, na kterých je tato disertační práce postavena. Jsou zde popsány základní pojmy týkající se spolehlivosti systémů, možné poruchy integrovaných obvodů a moderní technologie využité v obvodech FPGA. Následně jsou shrnuty základní principy a techniky pro zvýšení spolehlivosti systémů implementovaných uvnitř FPGA s využitím obvodové redundance a částečné dynamické rekonfigurace FPGA.

2.1 Spolehlivost systémů

Spolehlivost obecně vyjadřuje schopnost systému plnit požadovanou funkci při zachování hodnot stanovených provozních ukazatelů v daných mezích a v čase dle specifikovaných technických parametrů [38]. Spolehlivost chápána v tomto smyslu je komplexní vlastností (v anglické literatuře označována pojmem dependability), která je tvořena souborem čtyř dílčích vlastností systému označovaných jako *RAMS*[1] podle užívaných anglických názvů (uvedeno v závorkách):

- *Bezporuchovost* (reliability) je míra jistoty, že systém bude pracovat bez poruchy.
- *Dostupnost* (availability) je míra připravenosti systému poskytnout požadovanou funkci nebo servis.
- *Udržovatelnost* (maintainability) je schopnost systému podstoupit změny a opravy.
- *Bezpečnost* (safety) znamená absenci katastrofických důsledků poruchy systému na uživatele nebo prostředí.

Mezi další významné vlastnosti spolehlivosti mohou být zařazeny integrita (integrity) a věrohodnost (confidentiality). Integrita znamená absenci neočekávané nebo neautorizované změny dat v systému. Věrohodnost je nemožnost neoprávněného přístupu k informacím. Integrita, věrohodnost a také dostupnost jsou klíčové atributy pro bezpečnost (security) systému. Bezpečnostní systémy vyžadují oprávnění k přístupu, utajení informací a bezpečnost proti úmyslnému narušení [7].

[1] Analýza RAMS se v praxi často vypracovává při návrhu bezpečnostně kritických systémů. Analýza vyhodnocuje vliv možných poruch na funkci systému a ověřuje požadované spolehlivostní parametry [85].

Původně byla spolehlivost (ve smyslu komplexní vlastnosti systému) definována jako schopnost systému poskytnout službu, které lze odůvodněně věřit. Alternativní definice říká, že spolehlivost je schopnost systému vyhnout se takovému počtu selhání poskytované služby, které je častější a vážnější než je přípustné. Na základě těchto definic lze odvodit, že spolehlivému systému můžeme v odůvodněné míře důvěřovat do té doby, než selže vlivem poruchy [7]. Návrh spolehlivých systémů se proto zabývá způsoby, jak zabránit vzniku poruch nebo omezit vliv poruch v systému. Koncepce návrhu spolehlivých systémů je založena na vzájemné kombinaci a využití základních mechanismů [99]:

- *Prevence poruch* (fault avoidance) znamená využití takových prostředků a návrhových technik, které minimalizují pravděpodobnost vzniku poruch, jež mohou být ovlivněny během návrhu a vývoje systému. Prevence přirozených poruch znamená výběr spolehlivých (prověřených) nebo odolnějších součástek (např. speciální vytvrzené SRAM obvody FPGA), případně návrh robustnější konstrukce systému. Prevence návrhových a vývojových poruch znamená využití prověřených postupů, dodržování kvality vývoje, dodržení průmyslových standardů a hlavně verifikace návrhu. Statistické výsledky výzkumu provedeného v [31] ukazují, že čas strávený na verifikaci návrhu obvodů v FPGA se zvyšuje. To je důsledkem neustále se zvyšující komplexity návrhu (a tedy i jeho verifikace). Průmyslové standardy[2] vyžadují použití správných návrhových technik a implementaci takových bezpečnostních opatření, jež zajistí správnou funkci systému při zachování požadované spolehlivosti a bezpečnosti během předpokládané životnosti systému.

- *Odolnost proti poruchám* (fault tolerance) je důležitá vlastnost systému, od kterého se požaduje, aby byl schopný pokračovat ve své funkci navzdory přítomnosti poruch v systému. Odolnost proti poruchám může být implementována za pomoci technik pro detekci poruch a technik pro zotavení systému po poruše nebo maskování vlivu poruch na běh systému. Detekce poruch může probíhat preemptivně, např. při zapnutí systému nebo při plánované odstávce systému, nebo souběžně za běhu systému. Cílem zotavení systému je transformace stavu systému obsahujícího poruchy do bezporuchového stavu. Maskování poruch znamená kompenzaci funkce systému k poskytnutí správných výstupů, zatímco chyby jsou opraveny.

- *Odstranění poruch* (fault removal) znamená provádění preventivní nebo opravné údržby za běhu systému. Opravná údržba je provedena v případě zjištění chybového stavu, jenž může vést k poruše systému. Preventivní údržbu lze aplikovat pro odstranění takových poruch systému dříve, než způsobí chyby ve stavu systému (např. výměna komponenty systému). Odstranění poruch také probíhá během verifikace systému, jenž je jednou z nejdůležitějších fázi vývoje systémů.

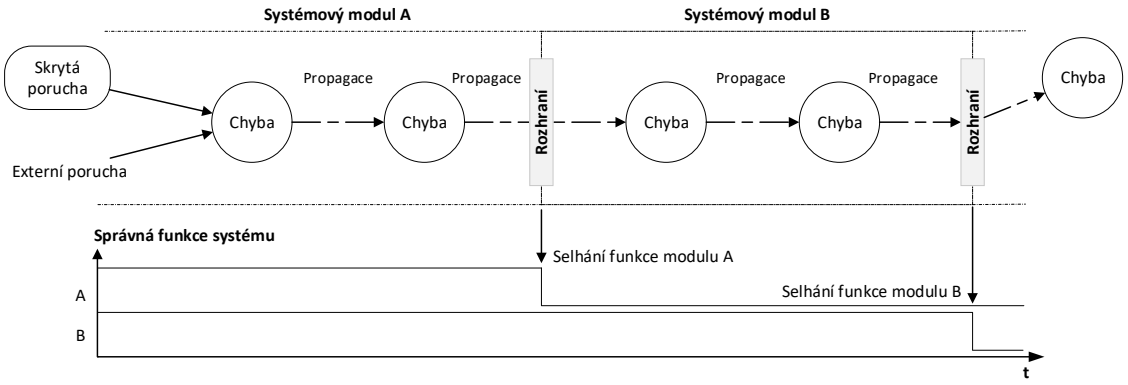
- *Předvídání poruch* (fault forecasting) znamená vyhodnocení chování systému s ohledem na aktivaci a četnost poruch. Vyhodnocení chování systému může být kvalitativní a kvantitativní. Cílem kvalitativního vyhodnocení je rozpoznat, klasifikovat a ohodnotit různé poruchové režimy na základě možných poruch komponent, podmínek a vlivu prostředí. Kvantitativní vyhodnocení má za cíl vyhodnocení četnosti poruch a odvození spolehlivostních parametrů pro návrh systému.

[2] Vývoj a certifikace komplexních digitálních obvodů pro civilní letecké systémy jsou řízeny standardem *DO-254* [33] vyžadovaným Evropskou agenturou pro bezpečnost letectví (EASA) i americkou Federální leteckou správou (FAA). Vývoj vesmírných systémů je např. v Evropě řízen standardem *ECSS* od evropské vesmírné agentury (ESA) [27].

2.1.1 Hrozby pro spolehlivost systému

Hrozby pro spolehlivost systému mohou být vyjádřeny za pomoci pojmů: porucha, chyba a selhání [7]. Prostřednictvím těchto pojmů mohou být popsány události mající vliv na správnou funkci systému. Obrázek 2.1 znázorňuje posloupnost událostí od vzniku poruchy, jejího šíření v podobě chyby, vedoucí až k selhání funkce systému.

Bezporuchový stav systému je opuštěn v okamžiku vzniku poruchy. Porucha je jev spočívající v ukončení schopnosti systému plnit požadovanou funkci podle zadaných technických podmínek. Projevy aktivní poruchy mohou být popsány jako chyba nebo selhání. Porucha, která se ještě nestihla projevit nebo jejíž přítomnost nebyla detekována, se označuje jako *skrytá porucha* (latent fault). Podle doby trvání vlivu mohou být poruchy označeny jako *dočasná porucha* (transient fault) nebo *trvalá porucha* (pernament fault). *Chyba* je odchylkou od správné a přesné hodnoty nebo výpočtu způsobující, že se nesprávná informace šíří uvnitř systému nebo vně. Chyby číslicových obvodů jsou většinou reprezentovány nesprávnou hodnotou výstupního signálu, který je produkován vadným obvodem. Porucha sekvenční logiky obvodu může mít za výsledek chybu dat uložených v registrech nebo paměti. Jako *selhání systému* může být označen přechod do stavu, do něhož systém přejde v důsledku poruchy nebo chyby. Selhání funkce číslicového obvodu je odchýlením v prováděné funkci obvodu nebo systému od specifikovaného chování. Selhání reprezentuje stav, ze kterého se systém může vrátit pouze provedením opravného mechanismu (např. nahrazením vadné součástky nebo rekonfigurací systému).



Obrázek 2.1: Posloupnost vzniku a šíření poruchy, chyby a selhání systému

2.1.2 Ukazatelé spolehlivosti

Spolehlivost je z pohledu návrhu celého systému komplexní vlastnost a nelze ji tedy jednoduše vyjádřit jedinou hodnotou. Proto se pro vyjádření spolehlivosti používají tzv. ukazatelé spolehlivosti. Lze říci, že ukazatele spolehlivosti jsou kvantitativním vyjádřením dílčích vlastností spolehlivosti systému.

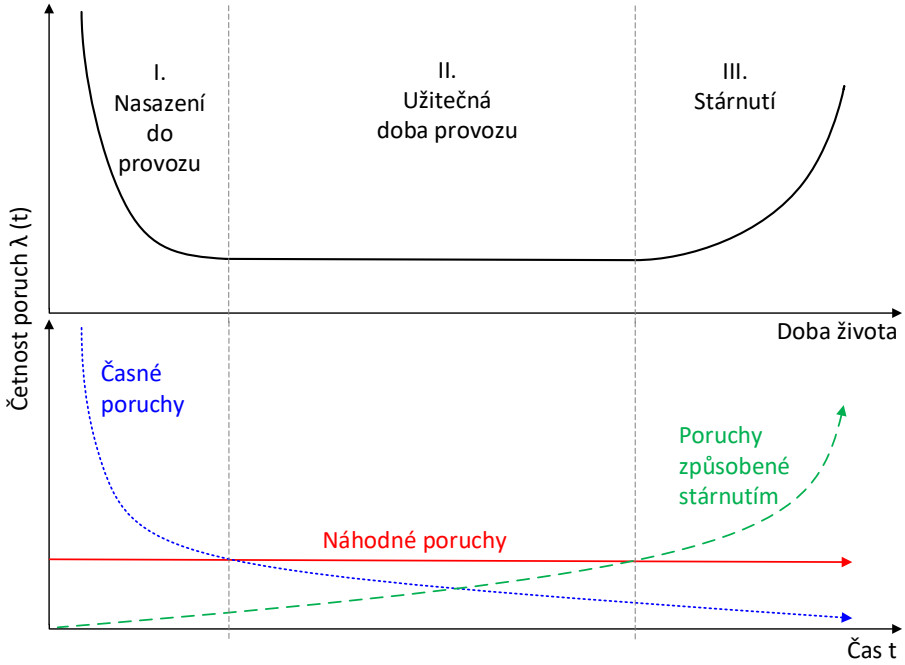
Doba od uvedení systému do provozu do jeho poruchy je základní veličinu sledovanou v teorii spolehlivosti. Spolehlivost systému je definována jako funkce $R(t)$ závislá na času, která udává pravděpodobnost toho, že systém nepřetržitě poběží bez poruchy v daném časovém intervalu $[0, t]$ (v případě, že systém pracuje správně v čase $t = 0$) [23].

Pouze minimum systémů je navrženo k nepřetržitému provozu bez jakéhokoliv přerušení nebo údržby. Proto je důležité také znát, jak dlouho může systém běžet bez přerušení.

Intenzita poruch je dána počtem očekávaných poruch v daném čase [23]. Intenzita poruch může být pro elektronické obvody a systémy zjednodušeně modelována za pomoci tzv. vanové křivky[3], znázorněné na obrázku 2.2.

Vanová křivka znázorňuje četnost poruch elektronického systému během předpokládané doby životnosti. Vanová křivka je složena ze tří částí: první křivka znázorňuje zvýšenou četnost poruch během počátečního období (např. vlivem častějších poruch součástek způsobených nedokonalostmi výrobního procesu), prostřední část znázorňuje užitečné období systému, třetí část ukazuje opětovný nárůst poruch vlivem stárnutí komponent systému. Předpokladem je, že během užitečné doby života systému má funkce intenzity poruch $z(t)$ konstantní hodnotu λ. Lze tedy říci, že spolehlivost systému exponenciálně klesá v závislosti s časem:

$$R(t) = e^{-\lambda t} \tag{2.1}$$



Obrázek 2.2: Vanová křivka četnosti poruch elektronického systému [85]

Pro komplexní systém, který nepoužívá redundanci, lze určit celkovou intenzitu poruch λ na základě součtu dílčích hodnot intenzit poruch λ_i všech komponent i, předpokládáme-li nezávislost jejich poruch (2.2). Tyto hodnoty jsou pro standardizované součástky často dostupné přímo od výrobců nebo v komerčních databázích součástek.

$$\lambda = \sum_{i=1}^{n} \lambda_i \tag{2.2}$$

[3]Vanová křivka je zjednodušený model poruch systému. U dnešních systémů se provádí analýza poruch za pomoci speciálních nástrojů, vyhodnocují se paramtery RAMS a spolehlivost systému na základě dekompozice systému a modelu poruch odpovídajícímu cílovému použití a prostředí.

Metody pro zvýšení odolnosti systému proti poruchám se zaměřují na užitečné období systému, což je nejdelší úsek jeho životnosti. Spolehlivostní opatření systému jsou navrhována na základě znalosti možného výskytu poruch systému v daném čase.

Ukazatel $MTTF$ (Mean Time To Failure) vyjadřuje předpokládanou dobu do první poruchy systému. Vztah mezi hodnotou MTTF a spolehlivostí $R(t)$ lze stanovit podle vzorce (2.3). Pokud má $R(t)$ spolehlivost exponenciální rozdělení, pak lze hodnotu MTTF vyjádřit za pomoci intenzity poruch λ (2.4).

$$MTTF = \int_0^\infty R(t)dt \tag{2.3}$$

$$MTTF = \frac{1}{\lambda} \tag{2.4}$$

Ukazatel MTTF se běžně uvádí v jednotkách FIT (Failures in Time), které vyjadřuje množství možných poruch za dobu provozu 10^9 hodin. Pokud je hodnota MTTF uvedena v hodinách, pak lze hodnotu FIT vyjádřit pomocí vzorce 2.5.

$$FIT = \frac{10^9}{MTTF} \tag{2.5}$$

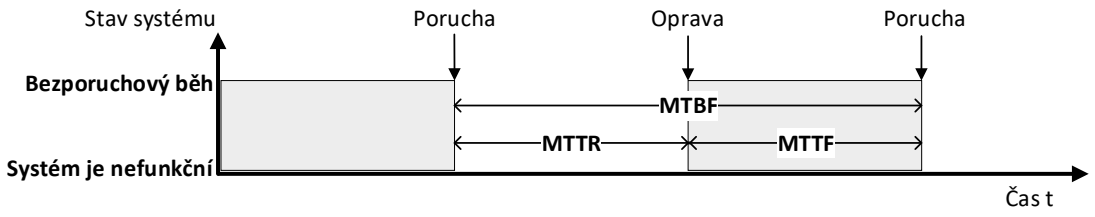
Dalším ukazatelem spolehlivosti, který je důležitý, pokud lze systém po vzniku poruchy opravit a pokračovat v jeho provozu, je střední doba do opravy $MTTR$ (Mean Time To Repair). Ukazatel MTTR může záviset na plánu údržby, umístění poruchy v systému nebo na použitém mechanismu pro obnovu stavu systému po vzniku poruchy.

MTTR se často uvádí ve smyslu četnosti oprav μ, která udává počet předpokládaných oprav za daný časový interval (2.6).

$$\mu = \frac{1}{MTTR} \tag{2.6}$$

Ukazatel $MTBF$ (Mean Time Between Failures) vyjadřuje střední dobu mezi poruchami. Předpokládáme-li, že se systém po opravě poruchy stane opět zcela bezporuchový, můžeme MTBF vyjádřit za pomoci součtu MTTF a MTTR (2.7). Vzájemný vztah mezi těmito spolehlivostními ukazateli je znázorněn i na obrázku 2.3.

$$MTBF = MTTF + MTTR \tag{2.7}$$



Obrázek 2.3: Ukazatelé spolehlivosti MTTF, MTTR a MTBF

Poměr mezi dobou bezporuchového běhu a součtem dob bezporuchového běhu a doby opravy systému udává celkovou dostupnost (availability) systému A. Hodnota A může být vypočtena za pomoci rovnice 2.8.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.8}$$

2.2 Poruchy číslicových systémů a obvodů FPGA

Současné číslicové systémy a programovatelné obvody FGPA obsahují milióny tranzistorů CMOS. Četnost poruch způsobených nedokonalostmi výrobního procesu a zastaráváním součástek se v dnešní době podařilo výrazně zredukovat díky použití nových výrobních technologií a důkladnému předcházení poruchám, které zahrnuje např. propracovaný vývojový cyklus, zavedení metodiky návrhu pro snadnou testovatelnost *DFT (Design for Testability)* [48], důkladnou funkční verifikaci, dodržování přesného výrobního procesu nebo přísné dodržování požadované kvality.

Nicméně během užitečné doby provozu číslicových obvodů, což je nejdelší fáze jejich životního cyklu, může dojít k dalším náhodným poruchám, které mohou mít přechodné nebo trvalé následky. Vzhledem k neustálému zvyšování počtu tranzistorů na čipu, zmenšování jeho plochy a zvyšování frekvence se některé integrované obvody a systémy na čipu *SoC (System on Chip)* staly mnohem náchylnější k poruchám, vznikajícím vlivem působení nepříznivého okolního prostředí a degradace elektronického materiálu [102], než dříve.

Poruchy číslicových systémů jsou většinou způsobeny chybami nebo nedokonalostmi výrobního procesu (chybějící kontakt, parazitní tranzistor), chybami a poškozením použitého polovodičového materiálu, dlouhodobým opotřebením součástek, degradací kontaktů a působením vlivu okolního prostředí. Vzniklé poruchy mohou být trvalé nebo dočasné:

- *Trvalé poruchy* (pernament faults) jsou v integrovaných obvodech nejčastěji způsobeny nedokonalostmi výrobního procesu. Destruktivní změny vedoucí k trvalé poruše elektronických součástek mohou být také způsobeny mechanickým, elektrickým nebo teplotním namáháním, mohou být následky opotřebení nebo stárnutí materiálu.

  Pro popis poruchových jevů v číslicových systémech se vzhledem k množství různých poruch, které mohou vzniknout, používají různé modely poruch. Mezi modely trvalých poruch lze zařadit např. poruchy typu stálá logická 0 (*stuck-at-0*) a stálá logická 1 (*stuck-at-1*), zkrat obvodu, přemostění signálu (*bridging fault*) nebo zpoždění v propagaci změny signálu (*delay fault*).

- *Přechodné poruchy* (transient faults), neboli dočasné poruchy, jsou v integrovaných obvodech nejčastěji způsobeny vlivem okolního prostředí. Doba trvání poruchového jevu je obyčejně velmi krátká. Tyto poruchy jsou většinou způsobeny přechodným dějem, po jehož odeznění může porucha zmizet. Nicméně přechodná porucha může způsobit chyby, které v číslicovém systému setrvávají nebo se šíří dále.

  Příčiny výskytu přechodných poruch v integrovaných obvodech mohou být různé, většinou jsou způsobeny vlivem vnějšího prostředí, kosmickým zářením, elektrostatickým výbojem, elektromagnetickým rušením, poruchami napájecího napětí nebo kolísáním okolní teploty.

Trvalé a přechodné poruchy způsobené působením kosmického záření na materiály uvnitř polovodičových součástek jsou dnes největší hrozbou pro většinu elektronických zařízení a systémů používaných na Zemi, ve vzduchu a ve vesmíru. Tato práce se dále soustředí právě na tyto poruchy, označované jednotným názvem SEE (Single Event Effects) poruchy.

2.2.1 Radiační prostředí okolo Země

Radiační prostředí ve vesmíru a okolo Země je tvořeno množstvím různých energetických částic, které se šíří v meziplanetárním prostoru anebo jsou zachyceny v magnetickém poli Země. Tyto energetické částice mohou být klasifikovány následovně:

1. Galaktické kosmické záření *GCR (Galactic Cosmic Ray)*, které obsahuje vysoce nabité částice, se šíří zpoza sluneční soustavy. Toto záření je tvořeno z 87% protony, 12% ionty helia (tj. alfa částice) a zbylé 1% jsou těžké ionty.

2. Sluneční kosmické záření *SCR (Sun Cosmic Ray)* je způsobeno sluneční aktivitou, během které jsou vyzářeny do prostoru nízko energetické částice pocházející ze slunečního větru a vysoce energetické částice emitované během slunečních erupcí. Záření SCR se skládá z elektronů, protonů, alfa částic, neutronů, těžkých iontů, gama paprsků a rentgenového záření (paprsků X).

3. Částice uvězněné v magnetickém poli Země, které vytvářejí tzv. Van Allenovy radiační pásy kolem Země. Tyto částice jsou většinou protony a elektrony. Van Allenovy radiační pásy jsou tvořeny vnějším a vnitřním pásem. Tyto pásy obklopují Zemi symetricky okolo její magnetické osy. Vnitřní pás obsahuje jak elektrony, tak protony, zatímco vnější pás je tvořen převážně elektrony.

   Na nízký orbit *LEO (Low Earth Orbit)*[4] je vyslána většina vesmírných družic. Nízký orbit je stíněn radiačními pásy, které poskytují vesmírným satelitům ochranu před kosmickým zářením. Nachází se zde také mezinárodní vesmírná stanice ISS. Vesmírné družice nebo rakety na geostacionárním orbitě *GEO (Geostationary Equatorial Orbit)*, a na vzdálenějších orbitech od Země, jsou vystaveny daleko vyšší koncentraci energetických částic.

Radiační prostředí z vesmíru prostupuje atmosférou Země. Interakcí energetických částic z primárního kosmického záření s jádry atomů prvků dusíku a kyslíku vzniká kaskáda tzv. sekundárního záření, jehož energetické částice dále působí na atomy prvků látek v atmosféře nebo se rozpadají. Přestože intenzita částic kosmického záření se snižuje s klesající nadmořskou výškou, existují místa na Zemi, kde může být koncentrace kosmického záření vyšší. Příkladem může být tzv. jiho-atlantická anomálie, kde se ve vyšších částech atmosféry se slabším magnetickým polem Země vyskytuje vyšší koncentrace kosmického záření.

2.2.2 Vliv kosmického záření na polovodičové součástky

Interakce energetických částic kosmického záření s atomy materiálů obsažených v polovodičových součástkách může mít vliv na jejich fyzikální a chemické vlastnosti. Polovodičové součástky jsou citlivé na *ionizující záření*[5].

Působení energetických částic může ovlivnit funkčnost součástek a způsobit jejich poruchu. Vzniklé poruchy mohou mít různé projevy i trvání, rozhodující je velikost energie částic, typ a doba vystavení působení záření.

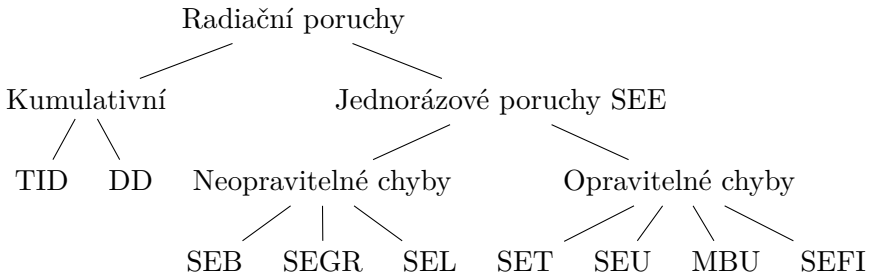
[4] Na nízký orbit byl v roce 2017 vyslán také první český nanosatelit *VZLUSAT-1*, typu *CubeCat*. Nanosatelit byl vypuštěn ve výšce 505 km nad mořem. Rozměry složeného nanosatelitu jsou 10cm × 10cm × 230cm. Řídící jednotka je ovládána 32-bitovým mikroprocesorem s jádrem ARM7. Jedním z hlavních cílů mise bylo ověřit nový koncept miniaturizovaného rentgenového dalekohledu [98].

[5] Ionizující záření může způsobit ionizaci látek, kdy interakce částice záření (fotonu nebo elektronu) s dostatečnou kinetickou energií a atomem prvku dané látky způsobí uvolnění elektronu z atomového obalu. K tomuto jevu dochází v případě, že je energie nabité částice větší než energie vazby atomu [114].

Na základě pozorovaných projevů poruch způsobených vlivem kosmického záření mohou být poruchové jevy rozděleny do dvou hlavních kategorií:

- Kumulativní poruchové jevy jsou způsobeny dlouhodobým působením kosmického záření. Do této kategorie patří poruchové jevy *TID (Total Ionising Dose)*, způsobující poruchy ionizací dielektrika, a *DD (Displacement Damage)*, způsobující poruchy v krystalové mřížce. Tyto jevy způsobující destruktivní poruchy.
- Jednorázové poruchové jevy, souhrnně označované jako poruchy *SEE (Single Event Effects)*. Do této kategorie patří poruchy, které vznikají v okamžiku jednorázového narušení materiálu v elektronických součástkách a přenosem své energie. Těmito poruchami se bude blíže zabývat následující kapitola.

Bližší rozdělení poruchových jevů způsobených vlivem kosmického záření je znázorněno diagramem na obrázku 2.4.

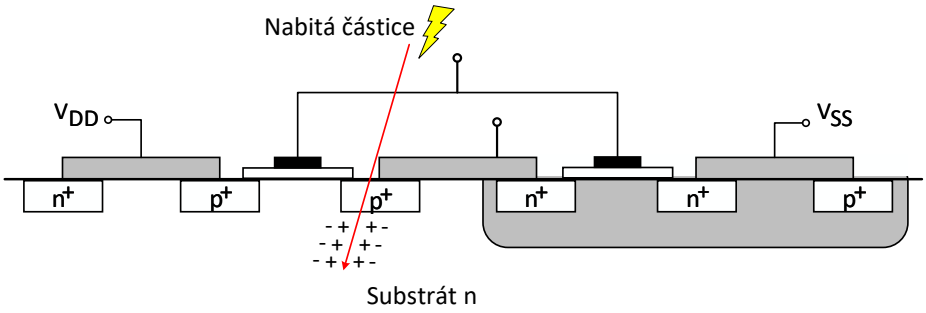


Obrázek 2.4: Radiační poruchy způsobené SEE jevy ovlivňující obvody FPGA

2.2.3 Jednorázové poruchy Single Event Effects

Poruchové jevy *SEE (Single Event Effects)* mohou způsobovat v polovodičových součástkách dvě kategorie chyb: tzv. měkké chyby (*soft errors*), které jsou opravitelné, a tzv. tvrdé chyby (*hard errors*), které jsou neopravitelné. Měkké chyby jsou důsledkem nedestruktivních poruch a často se projevují formou změny logické úrovně signálu v obvodě nebo stavu paměťového prvku součástky. Existence měkkých chyb může být přechodná nebo trvalá.

- *Single Event Upset (SEU)* je poruchový jev, který má za následek překlopení logického stavu jednoho bitu. Tato změna je výsledkem působení energetické částice, která způsobí vznik náboje v polovodičové součástce. Vznik poruchy typu SEU v polovodičové součástce je znázorněn na obrázku 2.5. Poruchy SEU ovlivňují paměťové buňky a registry v sekvenčních obvodech. Vzhledem k vysoké koncentraci těchto součástek v číslicových systémech jsou poruchy SEU největší hrozbou pro operační paměti, paměti cache a registry mikroprocesorů, klopné obvody, paměti a programovatelné logické bloky v obvodech FPGA. Tyto poruchy bývají většinou přechodné, což znamená, že stav ovlivněné paměťové buňky může být opraven nastavením správné logické hodnoty. Tyto poruchy mohou být snadno opraveny pomocí korekčních kódů [5].
- *Multiple Bit Upset (MBU)* je vícenásobný výskyt poruch typu SEU, který má za následek změnu více bitů v rámci bitového slova. Vícenásobné poruchy MBU jsou závažnější, jelikož způsobené chyby nelze jednoduše opravit pomocí jednoduchých



Obrázek 2.5: Vznik poruchy typu SEU v polovodičové součástce

korekčních kódů. Vícenásobné poruchy, které ovlivňují různé paměťové buňky, jsou označované jako *MCU (Multiple-Cell Upset)* [27].

- *Single Event Transient (SET)* je přechodné napěťové nebo proudové rušení ovlivňující kombinační hradla. SET nezpůsobuje přechod na výstupu zasaženého hradla, ale může se jako změna signálu propagovat skrz následující hradla a eventuálně způsobit překlopení logického stavu paměťové buňky (tedy poruchu typu SEU). Pravděpodobnost, že přechodný puls bude zachycen jako validní data v kombinační logice, roste lineárně s frekvencí obvodu (s větší frekvencí hran hodinového signálu se zvyšuje i pravděpodobnost vzniku poruchy typu SET) [70].

- *Single Event Functional Interrupt (SEFI)* je poruchový jev, který má za následek ztrátu nebo změnu funkce obvodu. Poruchy typu SEFI vznikají jako důsledek překlopení bitu (tedy vlivem SEU) v kritických systémových registrech, jenž mohou řídit např. operace a režimy, vykonávání programu, distribuci hodinového signálu v obvodech typu FPGA, pamětích typu SRAM a DRAM, nevolatilních pamětech Flash, nebo mikroprocesorech [10].

Zatímco měkké chyby nemají za následek trvalé poškození součástky, poruchové jevy způsobující trvalé poruchy jsou neopravitelné a destruktivní. Trvalé chyby se v polovodičových součástkách objevují často ve formě zkratu vedoucího k jejich zničení.

- *Single Event Latch-up (SEL)* je poruchový jev, při kterém se vytvářejí parazitické bipolární tyristory NPNP a PNPN v logice integrovaných obvodů CMOS, které mohou vést k vzniku zkratu a nadproudu. V případě, že nadproud není omezen, může dojít k přehřátí a destrukci ovlivněné součástky. Stav obvodu může být obnoven včasným restartem dodávky napájecího napětí [27].

- *Single Event Burnout (SEB)* je poruchový jev ohrožující bipolární tranzistory typu MOSFET, bipolární výkonové tranzistory a diody, jenž mají nastavený velký pracovní bod. Tento jev může způsobit zkrat a selhání součástky.

- *Single Event Gate Rupture (SEGR)* je poruchový jev ohrožující tranzistory typu MOSFET, při kterém může dojít k destruktivnímu vyhoření součástky vlivem zhroucení dielektrika.

2.3 Rekonfigurovatelné obvody FPGA

Moderní obvody FPGA se dnes rozlišují zejména použitou výrobní technologií, typem použité konfigurační paměti a způsobem návrhu programovatelné logické architektury na čipu.

Jeden z hlavních rozdílů, který rozlišuje obvody FPGA od různých výrobců, je technologie použitá pro vytvoření konfigurační paměti. Pro uchování vnitřní konfigurace obvodu FPGA jsou obecně používány technologie SRAM, Flash a tzv. antipojistky (Antifuse) [36] [56]. Hlavní rysy těchto technologií jsou porovnány v tabulce 2.1. V praxi má každý z typů těchto obvodů FPGA své specifické uplatnění.

- *SRAM FPGA* – Obvody FPGA založené na technologii SRAM jsou preferovány z důvodu jejich neomezené programovatelnosti a možnosti kompletní i částečné rekonfigurace FPGA za běhu systému. Tyto obvody jsou ovšem volatilní a tedy jejich konfigurace musí být do konfigurační paměti vždy při startu systému nahrána z externí paměti (např. EEPROM). Nutnost nahrání konfigurace může výrazně zpomalit start systému v porovnání s technologií Flash. Obvody SRAM FPGA mohou dosahovat díky nejnovější technologii CMOS vyšších výkonů než ostatní typy FPGA.
- *Flash FPGA* – Hlavní výhodou obvodů FPGA založených na technologii Flash je, že jsou programovatelné a zároveň nevolatilní. Technologie Flash je také méně citlivá na poruchy způsobené kosmickým zářením (v porovnání se SRAM). Problémem však může být to, že počet programovacích cyklů je omezen. Dříve se pro programovatelné nevolatilní FPGA používala technologie EEPROM, která byla později nahrazena technologií Flash z důvodu efektivnějšího využití plochy čipu [56].
- *Antipojistky* – FPGA založené na technologii antipojistek lze naprogramovat pouze jednou. Při programování jsou propojeny logické buňky a toto propojení je již neměnné. Jejich výhodou je, že zabírají výrazně menší plochu oproti ostatním typům obvodů FPGA. Logické buňky nejsou citlivé na poruchy typu SEU způsobené kosmickým zářením, díky čemuž jsou často využívány ve vesmírných aplikacích. Nicméně i v těchto obvodech se může objevit porucha typu SET. Výhradním výrobcem těchto obvodů je firma Microsemi (dříve Actel).

Vlastnost technologie FPGA	SRAM	Flash	Antipojistka
Přeprogramovatelnost	Ano	Ano	Ne
Počet přeprogramování	∞	10000	1
Rychlost naprogramování	Rychlá	Pomalá	–
Částečná dynamická rekonfigurace	Ano	Ne	Ne
Nevolatilní konfigurace	Ne	Ano	Ano
Kapacita konfigurační paměti	Vysoká	Střední	Nízká
Citlivost na SEE poruchy	Vysoká	Nízká	Velmi nízká

Tabulka 2.1: Porovnání hlavních rysů technologií FPGA [3] [56]

2.3.1 Technologie obvodů SRAM FPGA

Programovatelné obvody FPGA jsou složeny z konfigurovatelných logických buněk, propojovací sítě a vstupně/výstupních obvodů. Každý výrobce obvodů FPGA používá odlišnou

architekturu, technologii a terminologii. Firma Xilinx se zaměřuje primárně na obvody FPGA s konfigurační pamětí typu SRAM. Technologický vývoj obvodů FPGA od firmy Xilinx je znázorněn v tabulce 2.2. Tabulka chronologicky uvádí rok uvedení na trh a použitou výrobní technologii pro vybrané řady obvodů FPGA. Firma Xilinx vyrábí obvody FPGA v několika produkčních řadách. Obvody Spartan se orientují na nízko-nákladové aplikace. Obvody Virtex jsou nejvýkonnější a integrují specializované funkční bloky na jednom čipu. Řady Kintex a Artix vychází ze série Virtex a snaží se nabídnout kompromis mezi cenou, výkonem a spotřebovanou energií. Obvody FPGA Zynq nabízí systém na čipu, který integruje FPGA a jádro procesoru ARM. Nejnovější řada FPGA Versal nabízí adaptivní výpočetní platformu integrující programovatelnou logiku, více-jádrový procesor ARM a další specializované funkční jednotky v rámci jedné flexibilní architektury na čipu.

Rok	Technologie	Uvedené obvody FPGA na trh
1985	2.5um	XC2064 *(První FPGA od firmy Xilinx)*
1998	0.18	Spartan, Virtex
2005	90nm	Spartan-3E, Virtex-4
2006	65nm	Virtex-5
2009	45nm	Spartan-6, Virtex-6
2010	28nm	Virtex-7, Artix-7, Kintex-7, Zynq-7000
2013	20nm	Virtex Ulrascale, Kintex Ultrascale
2015	16nm	Virtex Ultrascale+, Kintex Ultrascale+, Zynq Ultrascale+
2017	28nm	Spartan-7
2019	7nm	Versal ACAP

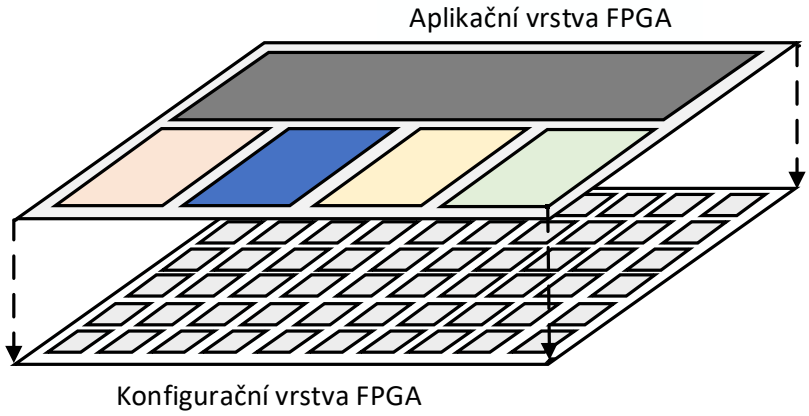
Tabulka 2.2: Technologický vývoj obvodů FPGA od firmy Xilinx [3][4][109]

Největší výhodou obvodů FPGA s konfigurační paměti SRAM je schopnost jejich částečné dynamické rekonfigurace. Částečná dynamická rekonfigurace vyžaduje, aby architektura daného obvodu FPGA umožňovala přepsání části konfigurační paměti a přeprogramování funkce příslušných logických bloků bez ovlivnění zbylé části systému za běhu celého obvodu. Obvody FPGA z rodiny Virtex byly první, které umožnili částečnou dynamickou rekonfiguraci. Novější obvody FPGA podporují částečnou dynamickou rekonfiguraci bez výjimky. Tento druh rekonfigurace je podporován také některými obvody FPGA od firem Intel nebo Atmel.

Základní struktura obvodů SRAM FPGA od firmy Xilinx

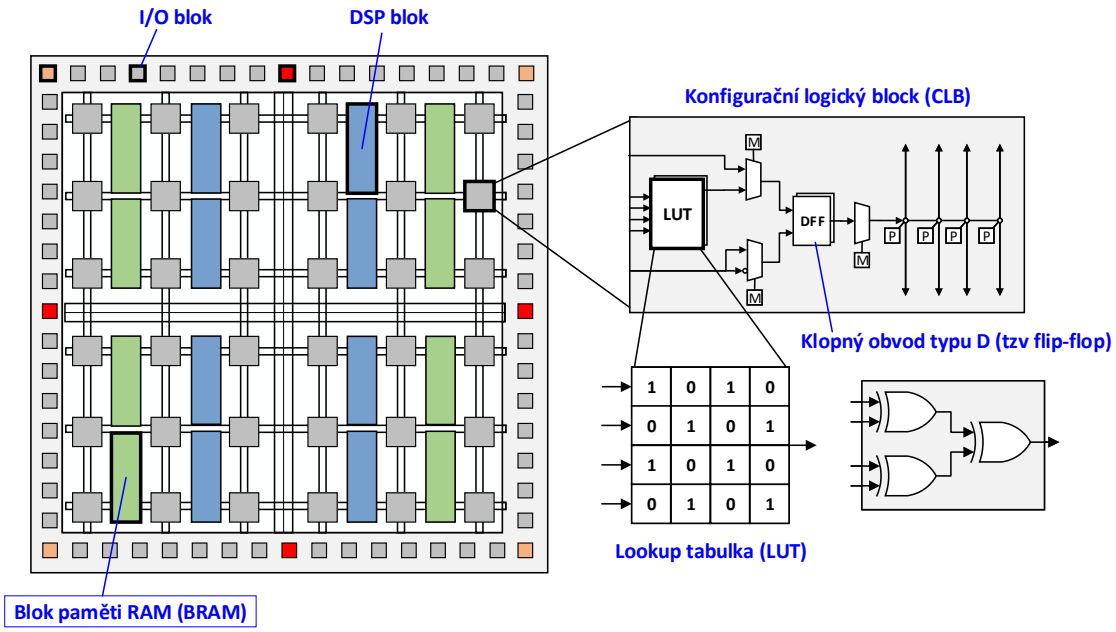
Architektura obvodů FPGA se skládá z konfigurační a aplikační vrstvy, jak je znázorněno na obrázku 2.6 [37]. Konfigurační vrstva se skládá z konfigurační pamětí, konfiguračních rozhraní (např. JTAG nebo SelectMAP) a řídicí logiky. Aplikační vrstva se skládá z logických, paměťových a vstupně-výstupních prvků, které implementují aplikační návrh.

Schéma struktury a logického rozložení obvodů SRAM FPGA od firmy Xilinx je znázorněno na obrázku 2.7. Aplikační logika je implementována konfigurovatelnými logickými bloky *CLB (Configuration Logic Block)*, které mohou realizovat dílčí sekvenční nebo kombinační obvody. *CLB* se skládá z několika tzv. *řezů* (*slice*). Každý řez v CLB může obsahovat více funkčních generátorů *LUT (Look-Up Table)*, paměťové prvky, logiku pro aritmetiku a multiplexory. Některé varianty řezu umožňují vytvoření např. distribuované paměti RAM nebo posuvného registru. Aplikační vrstva obsahuje také vstupně-výstupní bloky *IOB (Input/Output Block)*, které umožňují propojení FPGA s okolními zařízeními. Konfigurační



Obrázek 2.6: Konceptuální rozložení vrstev obvodů FPGA

bloky CLB jsou uspořádány do 2D matice, která je obklopena IOB porty. Programovatelná propojovací logika propojuje řádky a sloupce CLB. Různé parametry CLB pro vybrané architektury obvodů FPGA z rodiny Virtex jsou porovnány v tabulce 2.3.



Obrázek 2.7: Struktura obvodu FPGA

Aplikační vrstva může integrovat další doplňující funkční bloky, v závislosti na typu a architektuře obvodu FPGA. Obvody FPGA jsou často doplněny o speciální funkční bloky, které mohou zahrnovat: dvou-portové paměti *BRAM (Block RAM)*, dedikované násobičky, bloky pro zpracování číslicového signálu *DSP*, bloky pro řízení hodinového signálu *DCM (Digital Clock Manager)*, vestavěné procesory (např. procesor PowerPC v FPGA Virtex-II

Vlastnosti CLB	Virtex-4	Virtex-5	Virtex-6	Virtex-7	Virtex Ultrascale a Ultrascale+
Počet řezů (slices)	4	2	2	2	1
Počet vstupů LUT	4	6	6	6	6
Počet LUT	8	8	8	8	8
Počet klopných obvodů	8	8	16	16	16
Distribuovaná RAM [b]	64	256	256	256	512
Posunovací registr [b]	64	128	128	128	256

Tabulka 2.3: Porovnání parametrů CLB u řad FPGA Xilinx Virtex

Pro nebo procesor ARM-9 v FPGA ZYNQ) nebo vysokorychlostní Ethernetové rozhraní. Tabulka 2.4 porovnává parametry obvodů FPGA z rodiny Virtex od firmy Xilinx.

Obvody FPGA	Počet bloků Slice	Distribuovaná RAM [Kb]	Bloková RAM [Kb]	Počet bloků DSP
Virtex-4	5472 – 63168	86 – 1392	648 – 9936	32 – 512
Virtex-5	3120 – 51840	210 – 3420	936 – 18576	24 – 1056
Virtex-6	11640 – 118560	1045 – 8280	5616 – 38304	288 – 2016
Virtex-7	51000 – 305400	4338 – 17700	27000 – 67680	1120 – 3600
Virtex Ultrascale	358080 – 2532960	3900 – 28700	44300 – 132900	600 – 2880
Virtex Ultrascale+	394080 – 4085760	12000 – 58400	25300 – 94500	2280 – 12288

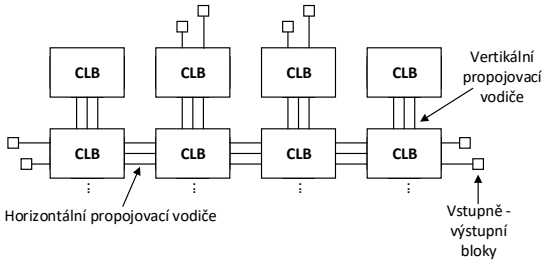
Tabulka 2.4: Porovnání parametrů vybraných obvodů FPGA od firmy Xilinx

Konfigurační rozhraní jsou uvnitř FPGA propojeny s registry a logikou řídicí proces konfigurace a s vlastní konfigurační pamětí. Konfigurační logika je přístupná také z aplikační vrstvy za pomoci vnitřního konfiguračního portu. Konfigurační paměť je organizována do rámců a dále do řádků (bližší detaily o struktuře konfigurační paměti FPGA jsou uvedeny v kapitole 2.3.2). Obvody FPGA mohou obsahovat také doplňující vestavěnou logiku pro opravu a detekci chyb v paměti, využívající korekčních kódů *ECC (Error Correction Code)* nebo kontrolního kódu *CRC (Cyclic Redundancy Check)*.

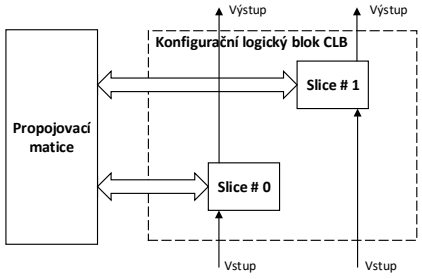
2.3.2 Architektura obvodu FPGA Xilinx Virtex-5

Architektura propojovací sítě v FPGA je znázorněna na obrázku 2.8. CLB jsou na tuto síť připojeny pomocí propojovacích bloků *CB (Connection Box)*. V místech průsečíku vodičů propojovací sítě jsou umístěny přepínací bloky *SB (Switch Box)*, které provádějí směrování signálů. Tato architektura se nazývá ostrůvková (*island*) a je použita ve většině obvodů SRAM FPGA [56].

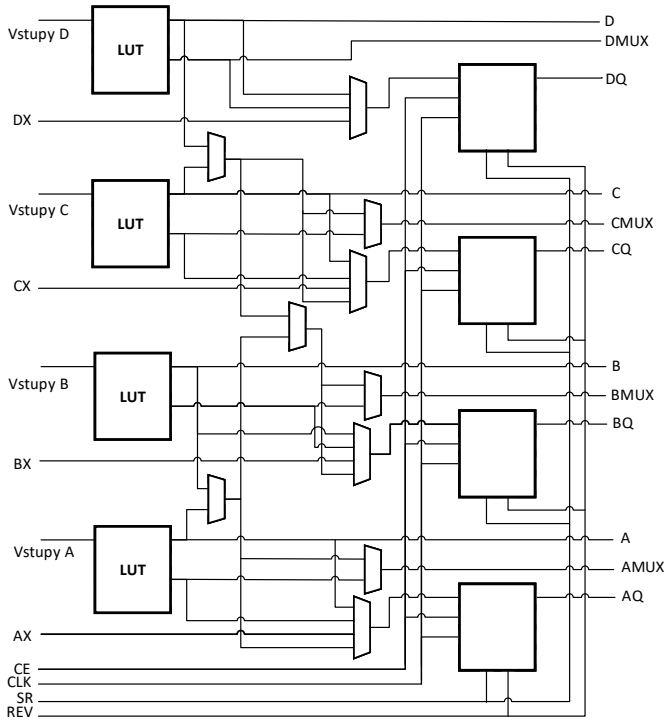
Konfigurační logický blok *CLB* zobrazený na obrázku 2.9 se u FPGA Virtex-5 [110] skládá ze dvou řezů (slice). Zjednodušené schéma architektury řezu je znázorněno na obrázku 2.10. Každý řez v CLB obsahuje čtyři funkční generátory LUT (Look-Up Table), čtyři paměťové prvky, logiku pro aritmetiku a multiplexory. Některé varianty řezu umožňují vytvoření např. distribuované paměti RAM nebo posuvného registru. LUT mohou pracovat se šesti nebo pěti vstupy a generovat dva výstupy. Paměťové prvky mohou být nakonfigurovány jako klopný obvod typu D reagující na náběžnou hranu signálu (flip-flop), případně jeho úroveň (latch).



Obrázek 2.8: Propojovací architektura FPGA



Obrázek 2.9: Struktura CLB



Obrázek 2.10: Architektura řezu (slice) v architektuře Xilinx Virtex-5

Konfigurace FPGA a konfigurační rozhraní

Způsob konfigurace FPGA můžeme klasifikovat podle jeho provedení několika způsoby. Rekonfigurace konfigurační paměti FPGA může být úplná nebo částečná. Dle okamžiku provedení můžeme rekonfiguraci rozdělit na statickou (konfigurace FPGA je přepsána během úplného zastavení systému) a dynamickou (probíhá za běhu systému). Rekonfigurace může být řízena interně vnitřním řadičem nebo z externího systému.

Kompletní rekonfigurace je nejjednodušší cestou pro nakonfigurování obvodu FPGA. Konfigurační data pro FPGA spolu s dalšími informacemi o cílovém obvodu FPGA jsou uložena v binárním souboru zvaném *bitstream*. Nahráním bitstreamu je celý obvod FPGA nakonfigurován, bez jakýkoliv dalších omezení. Tento způsob rekonfigurace je podporován všemi výrobci obvodů FPGA. Částečná dynamická rekonfigurace *PDR (Partial Dynamic Reconfiguration)* je schopnost obvodů FPGA s konfigurační paměti SRAM, kdy se místo

toho, aby se provedl restart obvodu FPGA a jeho kompletní rekonfigurace, nahrají nová konfigurační data a přeprogramuje se pouze specifická část konfigurace logiky FPGA, zatímco jsou ostatní části logiky FPGA stále v provozu [105].

Konfigurace obvodu FPGA Virtex-5 může být provedena pomocí různých konfiguračních rozhraní [111].

- *JTAG (IEEE standard 1149.1)* – Rozhraní *JTAG (Joint Test Action Group)* je primární konfigurační rozhraní používané především pro konfiguraci FPGA během fáze návrhu a testování.
- Sériové rozhraní – Sériové rozhraní umožňuje provést konfiguraci FPGA v režimu Master z paměti PROM. V režimu Slave lze přes sériové rozhraní provést konfiguraci z externího MCU nebo CPLD.
- *SelectMAP* – Rozhraní SelectMAP disponuje obousměrnou paralelní sběrnicí (8, 16 nebo 32 bitů) a může být tedy použito jak pro konfiguraci, tak pro zpětné čtení. Toto rohraní se často používá pro řízení PDR za pomoci externího systému (řadiče rekonfigurace implementovaném za pomoci procesoru nebo jiného FPGA, často odolnějšího proti poruchám).
- *ICAP* – Interní konfigurační rozhraní *ICAP (Internal Configuration Access Port)* je typicky používáno pro řízení PDR za pomoci interního řadiče rekonfigurace.
- *SPI* – Sériové rozhraní SPI lze využít pro automatickou konfiguraci konfigurační paměti FPGA v případě, že je interní paměť Flash PROM naprogramovaná.

Konfigurační rozhraní, která se využívají pro úplnou nebo částečnou rekonfiguraci FPGA, jsou porovnány v tabulce 2.5.

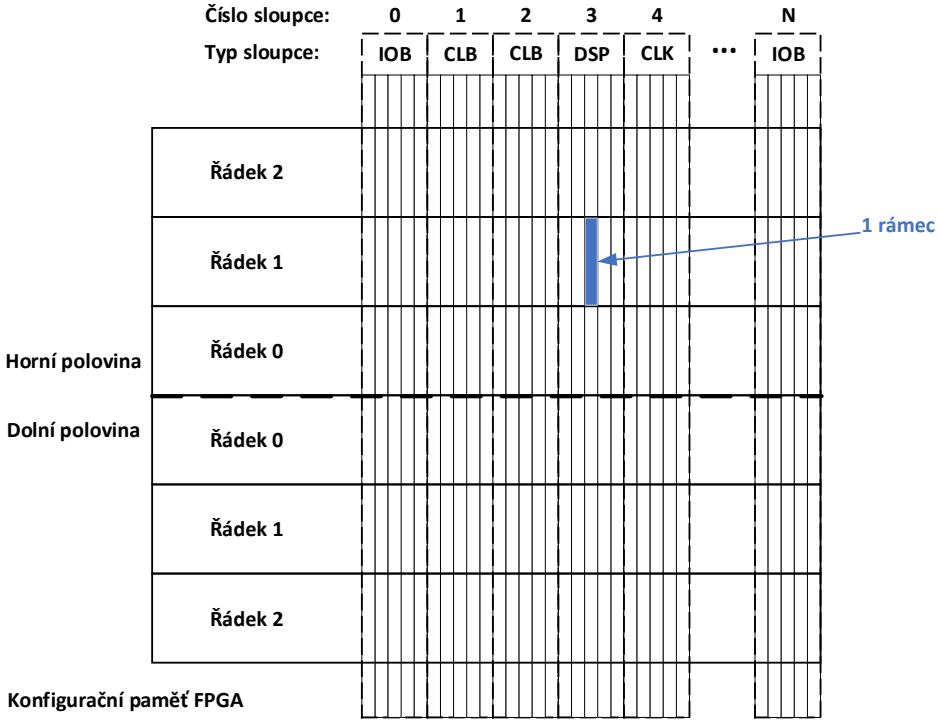
Konfigurační rozhraní	Datová šířka	Max. frekvence	Max. přenosová rychlost
JTAG	1 bit	66 MHz	66 Mbps
Serial mode	1 bit	100 MHz	100 Mbps
SelectMAP	32 bitů	100 MHz	3.2 Gbps
ICAP	32 bitů	100 MHz	3.2 Gbps

Tabulka 2.5: Porovnání parametrů konfiguračních rozhraní u rodiny FPGA Virtex [106]

Organizace konfigurační paměti FPGA

Schéma organizace konfigurační paměti FPGA je znázorněno na obrázku 2.11. Konfigurační paměť v FPGA je rozdělena na řádky a poté na sloupce. Obvody Xilinx Virtex-5 mohou mít až 12 řádků (v závislosti na velikosti konfigurační paměti). Střed konfigurační paměti je rozdělen na horní a dolní polovinu. Konfigurační rámec je nejmenší oblast, která může být rekonfigurována. Konfigurační rámec je zarovnán na celou výšku jednoho řádku konfigurační paměti a u FPGA Virtex-5 je rámec složen z 1312 konfiguračních bitů. Každý konfigurační rámec má unikátní 24-bitovou adresu *FAR*[6] *(Frame Address)*, která je rozdělena na 5 částí popsaných v tabulce 2.6. Každý sloupec může konfigurovat jeden typ programovatelných zdrojů v FPGA: IOB, blokovou paměť BRAM, CLB, DSP.

[6] Adresa FAR se používá při částečné dynamické rekonfiguraci, periodickém čištění konfigurační paměti a simulaci poruch SEU pro adresaci konfiguračních bitů FPGA.



Obrázek 2.11: Schéma organizace konfigurační paměti FPGA

Bitové pole	Význam části pole adresy FAR
b23 . . . b21	Typ bloku (většinou se jedná o konfigurační blok zdrojů FPGA)
b20	Příznak indikující horní nebo dolní polovinu konfigurační paměti FPGA
b19 . . . b15	Adresa řádku
b14 . . . b7	Adresa sloupce
b6 . . . b0	Adresa konfiguračního rámce uvnitř sloupce)

Tabulka 2.6: Adresa konfiguračního rámce FAR u obvodů FPGA Xilinx Virtex

2.3.3 Implementace návrhu číslicového systému do FPGA

Implementace návrhu číslicového systému do obvodu FPGA se sestává z několika fází, které lze z hlediska vývojového cyklu rozdělit na návrh, implementaci, verifikaci a konfiguraci FPGA. Celý návrhový proces[7] včetně dílčích kroků vedoucích k vytvoření cílové konfigurace pro konkrétní obvod FPGA lze popsat pomocí následujících kroků:

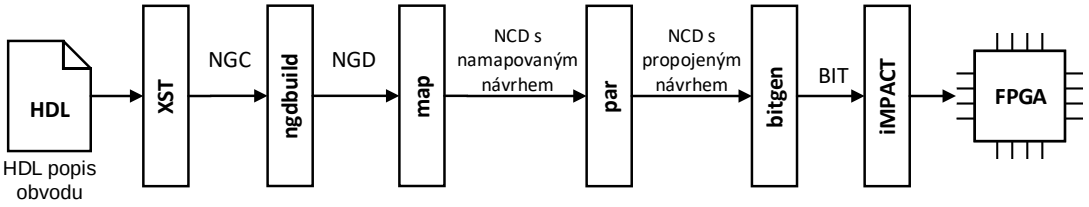
1. *Proces návrhu* – Prvním krokem návrhu číslicového systému je specifikace jeho funkce. Specifikace by měla pokrýt všechny funkční aspekty, rozhraní, parametry a omezení budoucího systému. Systém může být popsán pomocí jazyku *HDL (Hardware Description Language)* pro popis obvodů. Nejpoužívanější jazyky HDL jsou VHDL a Verilog. (Případně lze využít některého z vysokoúrovňových jazyků pro logickou syn-

[7]Způsob návrhu a implementace rekonfigurovatelného návrhu číslicového systému do FPGA s využitím specifických nástrojů od firmy Xilinx bude blíže popsán v kapitole 5.2.

tézu.) Popis obvodů číslicového návrhu může být proveden na různých úrovních abstrakce: na úrovni hradel (strukturní schématická úroveň), na úrovni RTL (úroveň mezi-registrových přenosů) a na úrovni chování obvodů (behaviorální popis).

2. *Proces implementace návrhu* – Během implementace návrhu je vstupní popis systému převeden za pomoci kroků logické syntézy, mapování, rozmístění a propojení do popisu systému umožňujícího konfiguraci konkrétního cílového obvodu FPGA. Proces logické syntézy popisu obvodu na úrovni RTL, mapování, rozmístění a propojení logických prvků je znázorněn na obrázku 2.13.

   Důležitým aspektem návrhového postupu jsou zvolené vývojové nástroje. V této disertační práci jsou popsány vývojové nástroje poskytnuté firmou Xilinx v rámci vývojového prostředí *ISE Design Suite*[8]. Obrázek 2.12 znázorňuje návaznost vývojových nástrojů od firmy Xilinx.



Obrázek 2.12: Návrhový proces s využitím nástrojů od firmy Xilinx

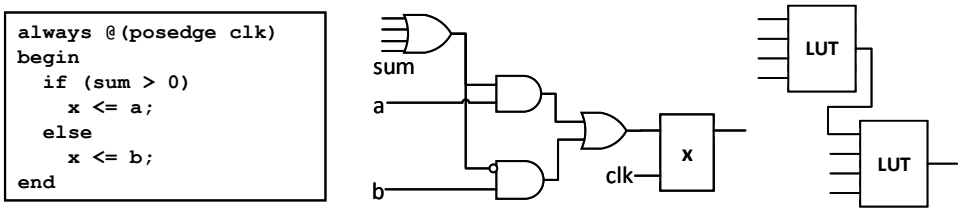
   2.1. *Logická syntéza (logic synthesis)* – Proces konverze popisu systému z vyšší úrovně abstrakce, tedy z popisu obvodů v jazyce HDL, na nižší úroveň, do seznamu základních logických prvků a jejich vzájemného propojení (tzv. *netlist*) [3]. Spolu se vstupním popisem logiky obvodů systému dodává návrhář také sadu uživatelských omezení (tzv. *constrains*), které umožňují např. propojení logických signálů návrhu se vstupně-výstupními bloky daného obvodu FPGA nebo definici časových kritérií na funkci obvodů. Nástroj od firmy Xilinx se jmenuje *XST*, jeho vstupem je popis obvodu v HDL a jeho výstupem je netlist ve formátu *NGC (Native Generic Compiler)*.

   2.2. *Mapování (mapping)* – Proces konverze technologicky nezávislého netlistu do netlistu využívajícího struktury cílové technologie. Před provedením mapování, jsou za pomocí nástroje *NGDbuild*, zpracovány všechny vstupní netlisty ve formátu NGC. Výstupem je soubor *NGD (Native Generic Database)*, který popisuje navržený číslicový systém s využitím primitiv FPGA. Současně jsou připojeny informace o uživatelských omezeních ze souboru *UCF (User Constraint File)*. Proces mapování je proveden pomocí nástroje *map*. Výstupem mapování je soubor *NCD (Native Circuit Description)*.

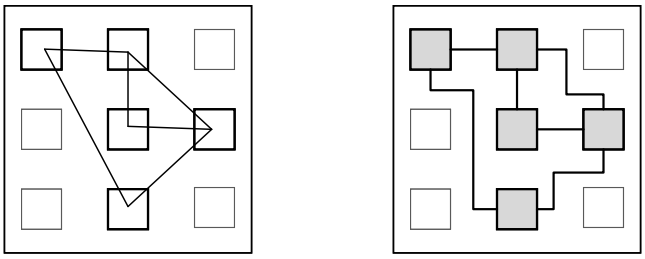
   2.3. *Rozmístění a propojení (place and route)* – Proces, při kterém jsou logické prvky číslicového návrhu, namapovaného na struktury cílové technologie, rozmístěny v rámci plochy obvodu FPGA a vzájemně propojeny. Během tohoto procesu je

[8] V praxi lze využít různé vývojové nástroje třetích stran, případně open-source nástroje, jenž podporují zvolený obvod FPGA. Pro návrh obvodů pro novější obvody FPGA od firmy Xilinx je nutné využít novější vývojové prostředí *Vivado*.

snahou optimalizovat propojení tak, aby bylo dosaženo minimálního zpoždění ve vodičích. Proces rozmístění a propojení je proveden pomocí nástroje *par*. Výstupem je opět soubor NCD.



(a) Popis na úrovni RTL (b) Netlist na úrovni hradel (c) Netlist na úrovni LUT



(d) Rozmístění logických prvků (e) Propojení logických prvků

Obrázek 2.13: Proces syntézy, mapování, rozmístění a propojení logických prvků obvodu

3. *Proces verifikace návrhu* – Během verifikace návrhu se ověřuje správnost implementovaného systému a dodržení časových parametrů systému. Verifikace popisu obvodů v HDL se provádí nejdříve za pomocí simulace RTL kódu, díky čemuž lze provést rychlé ověření správné funkce. Dále lze návrh systému verifikovat pomocí simulace na úrovni hradel (po dokončení syntézy návrhu) a za pomocí simulace finální implementace na úrovni hradel s reálnými zpožděními (po dokončení procesu rozmístění a propojení).

4. *Proces vygenerování konfigurace* – Cílovou konfiguraci pro obvod FPGA, ve formátu binárního konfiguračního souboru, lze vygenerovat pomocí nástroje *bitgen*. Binární konfigurační soubor se nazývá *bitstream*.

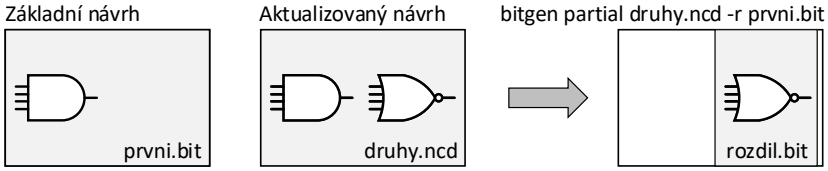
5. *Proces konfigurace FPGA* – Naprogramování obvodu FPGA je poslední fází návrhového postupu. Pro konfiguraci FPGA lze využít některého z konfiguračních rozhraní, popsaných v kapitole 2.3.2. Pro naprogramování obvodu FPGA z PC lze použít nástroj *iMPACT*.

2.3.4 Částečná dynamická rekonfigurace

Schopnost PDR obvodů FPGA s konfigurační pamětí SRAM rozšiřuje použitelnost těchto obvodů v systémech, kde je možnost výměny, aktualizace nebo rozšíření funkcionality za běhu systému velkým přínosem. Tyto možnosti umožnily vznik nových výzkumných směrů, které se začaly zabývat způsobem využitím PDR v mnoha oblastech a aplikacích. Obvody

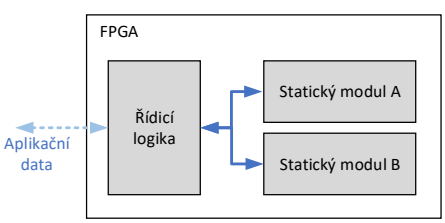
FPGA od firmy Xilinx podporují následující dvě techniky provedení částečné dynamické rekonfigurace návrhu v FPGA:

a) Rozdílová částečná rekonfigurace (*Difference-based PDR*) – Tento způsob rekonfigurace umožňuje provést malé změny v programovatelné logice za běhu FPGA s využitím konfiguračního bitstreamu, kterým se naprogramují pouze změny rozdílné mezi původním a pozměněným návrhem. Díky tomu je provedení změn v programovatelné logice daleko rychlejší než u ostatních způsobů rekonfigurace. Za pomoci této techniky lze změnit např. funkci LUT, režim vstupně-výstupních bloků nebo obsah pamětí BRAM. V případě, že je nutná změna ve vnitřním propojení komponent, není tento způsob rekonfigurace doporučen [105]. Na obrázku 2.14 je znázorněn příklad vytvoření rozdílové konfigurace pro aktualizaci logiky v FPGA.

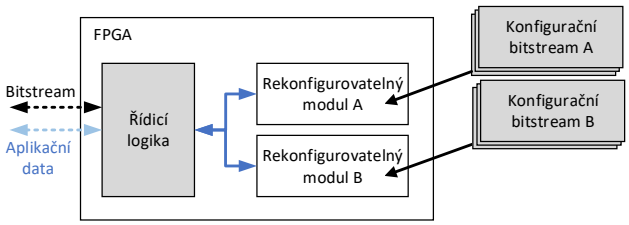


Obrázek 2.14: Rozdílová částečná rekonfigurace logiky v FPGA

b) Modulární částečná rekonfigurace (*Module-based PDR*) – Rekonfigurovatelný návrh FPGA může být rozdělen na statickou a dynamickou část. Statická část obsahuje komponenty, které mají zůstat aktivní po celou dobu běhu systému (např. interní řadič rekonfigurace). Dynamická část návrhu je tvořena rekonfigurovatelnými regiony *RR (Reconfigurable Region)*, jejichž velikost a tvar je definován návrhářem. Takových regionů může být v konfigurační paměti FPGA vymezeno více. Pro každý RR musí být implementován rekonfigurovatelný modul *PRM (Partial Reconfiguration Module)* obsahující část logiky obvodů systému, která má být rekonfigurovatelná. Pro jeden RR může být implementováno více PRM s odlišnou funkcí. Obrázky 2.15 a 2.16 znázorňují rozdíl mezi statickým návrhem v FPGA a dynamickým návrhem.



Obrázek 2.15: Statický návrh



Obrázek 2.16: Modulární dynamický návrh

Jelikož se tato disertační práce zabývá problematikou synchronizace stavu rekonfigurovatelných modulů v architektuře TMR, bude v dalším textu pod pojmem částečná dynamická rekonfigurace vždy uvažována modulární částečná rekonfigurace. Využití modulární PDR může být rozděleno do několika skupin:

- *Přepínání funkce systému za běhu* – PDR může být využita pro dynamické přepínání funkce rekonfigurovatelného modulu systému, který může být překonfigurován

na některou z předem definovaných funkčních variant. Důvodem pro rekonfiguraci části systému může být také nedostatek programovatelné logiky pro implementaci kompletního návrhu systému uvnitř FPGA.

- *Rekonfigurace systému pro úsporu spotřeby energie* – Největší úspory spotřeby energie lze dosáhnout, když se neaktivní moduly PRM systému rekonfigurují prázdnou konfigurací (*blank*). V tomto stavu má logika PRM minimální statický příkon (*leakage power*). Podle výzkumu autorů [61] se tato technika vyplatí, pokud je úspora energie větší, jak spotřebovaná energie rekonfigurací. Klíčem k minimalizaci spotřeby energie je rychlost provedení rekonfigurace.

- *Rekonfigurovatelné provádění úloh v HW* – Rekonfigurovatelné PRM lze využít pro provádění dedikovaných operací, které mohou být považovány za úlohy prováděné v HW ve smyslu tradičních programových procesů nebo vláken implementovaných v prostředí operačního systému. Autoři [45] [46] [40] [1] se zabývají návrhem operačního systému pracujícího v reálném čase pro spolehlivý rekonfigurovatelý systém *R3TOS (Reliable Reconfigurable Real-Time Operating System)*, který by umožnil řízení a plánování provádění úloh v HW.

- *Rekonfigurovatelný multi-procesorový systém na čipu* – Multi-procesorový systém lze v prostředí FPGA snadno vytvořit přidáním několika instancí jader procesorů a jejich vzájemným propojením. Výzkum prezentovaný v [39] se zabývá vytvořením rekonfigurovatelného multi-procesorového systému, ve kterém je každý procesor umístěn do dedikovaného rekonfigurovatelného modulu. Základní úvahou je, že v případě poruchy jednoho procesoru je rychlejší použít záložní procesor než čekat na opravu provedenou při periodickém čištění konfigurační paměti. Záložní procesory jsou udržovány provozuschopné za pomoci periodického čištění prováděného na pozadí.

- *Adaptivní výpočetní systémy* – Adaptivní systémy jsou schopné adaptovat svou funkci a strukturu měnícím se provozním podmínkám a okolnímu prostředí, optimalizačním cílům a fyzickým omezením [51]. Díky PDR lze velmi snadno změnit funkci obvodů implementovaných uvnitř FPGA v případě nalezení chyby v návrhu nebo při pozdější změně specifikace. PDR je také výhodná pro systémy s odloženou údržbou a dlouhou dobou mise, jako jsou např. vesmírné satelity nebo sondy [30]. Dalším příkladem může být programově definovaný radiový systém (SRD *Software Defined Radio*). SRD může podporovat množství různých komunikačních protokolů a přizpůsobit svou funkci a využité komunikační moduly zvolené šířce přenosového pásma [51].

- *Oprava poruch* – V systémech odolných proti poruchám může být rekonfigurace využita také pro opravu stavu systému v případě výskytu poruch v konfigurační paměti FPGA nebo aplikační logice způsobených působením kosmického záření. Přechodné poruchy mohou být odstraněny z konfigurační paměti za pomoci jejího periodického přepisování. Pro opravu stavu systému z trvalé poruchy v konfigurační paměti FPGA může být využita technika přenesení konfigurace PRM do nevyužité části FPGA.

2.3.5 Model poruch obvodu SRAM FPGA

Poruchy, které mohou vzniknout působením SEE jevů na konfigurační a logické vrstvě obvodů SRAM FPGA mohou mít přechodný nebo trvalý charakter. Vzniklé chyby nejsou destruktivní, ale změnou logické hodnoty konfiguračního bitu (porucha typu SEU) nebo

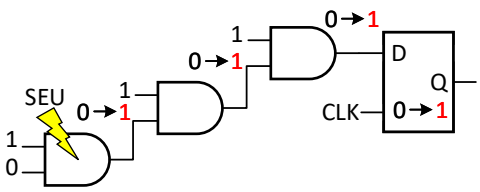
logické úrovně signálu (porucha typu SET) mohou ovlivnit správnost prováděné funkce obvodu. Nejčastějším typem poruchových jevů, které můžeme pozorovat u obvodů SRAM FPGA, jsou poruchy SEU a SET. Důsledkem těchto poruch může dojít také k poruchovému jevu SEFI, který označuje změnu funkce FPGA způsobenou chybou. Chyby způsobené jevy SEE v obvodech SRAM FPGA mohou být následující [5]:

- Poruchy v aplikační vrstvě FPGA:
  - *Překlopení bitu v logických hradlech uvnitř kombinační logiky* – Porucha SEU může ovlivnit kombinační logiku a způsobit přechodnou poruchu v logických hradlech. Způsobená porucha se může dále propagovat, nepřímo ovlivnit sekvenční část obvodu a způsobit překlopení bitu. Způsob vzniku SEU v logických hradlech je znázorněn na obrázku 2.17a.
  - *Překlopení bitu v aplikačních klopných obvodech a paměťových buňkách* – Porucha SEU může přímo ovlivnit obsah klopných obvodů a pamětí. Obsah těchto paměťových elementů zůstane chybný do té doby, než je příslušný překlopený bit opraven za pomoci technik pro detekci a korekci chyb.
- Poruchy v konfigurační vrstvě FPGA:
  - *Propojovací chyby* – Propojovací chyby vznikají v programovatelných propojovacích bodech PIP (Programmable interconnect points), multiplexorech a vyrovnávacích pamětech, které spolu vytváří programovatelnou propojovací síť uvnitř FPGA. Programovatelná obvodová logika je tvořena více jak 80% všech tranzistorů použitých v obvodě FPGA. Důsledky poruch typu SEU v propojovací logice jsou znázorněny na obrázcích 2.17c a 2.17d.
  - *Překlopení bitu konfigurujícího logickou funkci LUT* – Vznik poruchy SEU v pravdivostní tabulce LUT může změnit nakonfigurovanou logickou funkci. Příklad poruchy SEU v tabulce LUT je znázorněn na obrázku 2.17b.
  - *Překlopení bitu s řídicí funkcí* – Překlopení bitu s řídicí funkcí uvnitř bloků CLB nebo vstupně-vystupních bloků IOB může výrazně narušit funkci implementovaného obvodu. Řídicí bity uvnitř CLB např. určují, zda se daná LUT chová jako vyhledávací tabulka LUT, dvou-portová paměť nebo posunovací registr. Dalším příkladem mohou být konfigurační bity, které řídí propojení hodinového signálu uvnitř obvodu FPGA.

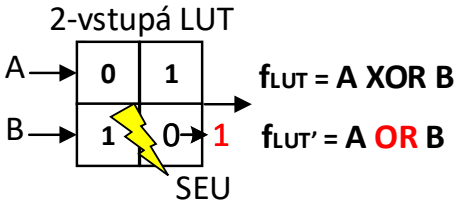
Tabulka 2.7 uvádí hodnoty výskytu poruch typu SEFI a SEU pro obvod FPGA Xilinx Virtex-4QV SX55 určené za pomoci modelu CREME96. Nadmořská výška odpovídající LEO byla stanovena na 800 km a nadmořská výška pro GEO byla použita 36000 km. Odhadnuté hodnoty byly estimovány pro všechny konfigurační bity použitého obvodu FPGA.

Stanovení vlivu SEE poruch na spolehlivost zařízení

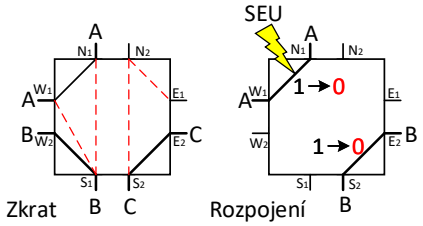
Při stanovení vlivu SEE poruch způsobených kosmickým zářením se často používá parametr *SER (Soft-Error Rate)*. *SER* udává četnost, s jakou měkké poruchy vznikají pro dané zařízení a prostředí. Hodnota SER může být vyjádřena pro známé nebo referenční prostředí za pomoci FIT, kdy jednotka FIT odpovídá jedné poruše při miliardě hodin provozu zařízení (tzn. jedna porucha za 114155 let).



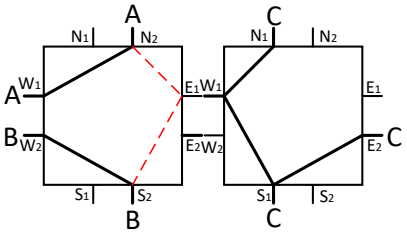
(a) SEU a překlopení stavu klopného obvodu



(b) SEU v LUT ovlivňující logickou funkci



(c) SEU ovlivňující logické propojení



(d) SEU způsobující zkrat v logickém propojení

Obrázek 2.17: Poruchy způsobené vlivem SEU v obvodech SRAM FPGA

Poruchy detekované v FPGA	LEO	GEO
Funkční porucha SEFI [roky/porucha]	36	103
Sekvenční klopné obvody (FF) [porucha/den]	0.0702	0.0387
Paměť Block RAM [porucha/den]	4.05	4.49
Konfigurační paměť [porucha/den]	7.56	4.28

Tabulka 2.7: Příklad hodnot výskytu poruch SEFI a SEU na orbitě u LEO a GEO [82]

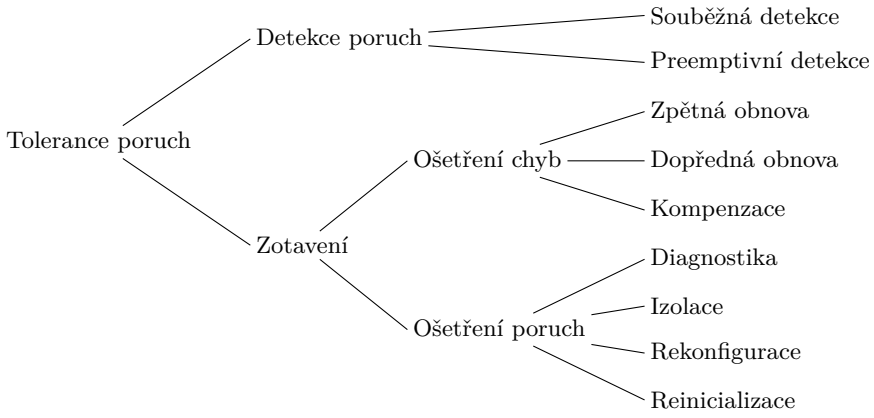
Hodnoty SER se pohybují u elektronických zařízení mezi 100 a 100000 FIT (cca. jedna porucha za rok) [70]. Citlivost polovodičových pamětí je často udávána v jednotkách FIT/Mb nebo *FIT na zařízení*. Jednotka FIT je dána počtem poruch za 10^9 hodin. Hodnota FIT je odhadnuta na základě simulace nebo experimentálního měření. Hodnota SER může být vypočtena na základě hodnoty citlivosti zařízení na SEE poruchy, kterou lze vyjádřit pomocí parametru σ (*cross-section*), jenž lze vypočítat jako poměr počtu pozorovaných chyb n v zařízení působením toku N částic na cm^2, jako $\sigma = \frac{n}{N}$. Za pomoci hodnoty částicového toku ϕ (*particle flux*) v reálném prostředí, vyjádřeném v n/cm^2/h, může být hodnota FIT vypočtena pomocí rovnice 2.9. Hodnota $MTTF$ může být určena na základě vzorce 2.5.

$$FIT = \frac{n}{N} \times \phi \times 10^9 \tag{2.9}$$

2.4 Koncepce systémů odolných proti poruchám

Rozdělení nejpoužívanějších metod v návrhu systémů odolných proti poruchám je znázorněno na obrázku 2.18. Základními schopnostmi těchto systémů je detekce poruch a zotavení po vzniku poruchy. Detekce poruch může probíhat souběžně s provozem systému nebo pre-

emptivně, kdy je provoz systému zastaven, což umožňuje detekci trvalých a skrytých poruch. Jakmile je porucha detekována, musí být zamezeno šíření chyb způsobených poruchou a dalšímu ovlivnění provozu systému. Mimo to je nezbytné přivést stav systému do původního bezporuchového stavu dříve, než se porucha začne šířit v systému (tzn. aktivuje jinou poruchu) a předejít situaci, kdy nebude možné tolerovat množství právě vzniklých poruch. Mechanismy pro zotavení systému po vzniku poruchy lze rozdělit na dvě části, metody ošetřující chyby a metody pro ošetření jejich příčin, tedy poruch. Klasické techniky pro ošetření chyb jsou zpětná obnova (*rollback*) a dopředná obnova (*roll-forward*). Základním principem zpětné obnovy je pravidelné ukládání zálohy stavu systému (*checkpoint*) v čase, kterou lze použít pro pozdější obnovu stavu systému.



Obrázek 2.18: Taxonomie technik typických pro systémy odolné proti poruchám

Redundance je klíčovým prvkem pro návrh systémů odolných proti poruchám. Redundance je nejjednodušší technikou používanou pro zvýšení spolehlivosti v systému umožňující také detekci a maskování poruch. Redundance zavádí záměrné zálohování systémových prostředků, jejichž použití by bylo zbytečné, kdyby všechny ostatní části systému pracovaly správně. Redundance může být aplikována ve třech různých formách zaměřujících se na zálohování fyzických prostředků, zálohování informací a opakování provedených operací. Na základě tohoto rozdělení lze typy redundance členit následovně:

- *Obvodová redundance* – Redundance na úrovni jednotlivých obvodů, komponent a systémů se často používá pro splnění systémových požadavků na dostupnost a spolehlivost a to i v případě, že komponenty tvořící systém nesplňují požadovanou spolehlivost. Redundance systémů může být použita v bezpečnostně kritických systémech i v případě, že samotný systém je dostatečně spolehlivý. Důvodem může být požadavek na odolnost systému vůči poruchám.

- *Informační redundance* – Zabezpečení dat systému pomocí informační redundance je založeno na principu zakódování přenášené informace umožňující zjistit nebo i opravit chybu vzniklou při přenosu vlivem technické nedokonalosti nebo poruchy přenosového kanálu. Zakódování pro zabezpečení informace při přenosu zvyšuje objem přenášených dat. Informační redundance se používá v zabezpečovacích kódech pro zajištění integrity a správnosti dat v pamětech (např. BCH, R-S) a přenosových kanálech (např. parita, kontrolní součet CRC). Některé kódy umožňují kromě detekce poruchy také její opravu.

- *Časová redundance* – Časová redundance spočívá ve dvou nebo vícenásobném opakování téže operace, výpočtu nebo přenosu dat a porovnání všech obdržených výsledků. Časová redundance může být snadno implementována programově, opakováním instrukcí nebo paralelním výpočtem ve více výpočetních vláknech.

2.5 Využití obvodové redundance pro detekci a maskování poruch

Obvodová redundance se velmi často používá v různých bezpečnostně-kritických systémech a systémech odolných proti poruchám. Hlavní výhodou obvodové redundance je schopnost detekce a maskování poruchy v systému. V případě zdvojení systému a porovnání jeho redundantních výstupů lze bezpečně rozpoznat poruchu v systému. V případě triplikace systému a výběru správných výstupů na základě majoritního volení lze dosáhnout schopnosti maskování poruch v systému. Základní techniky založené na použití obvodové redundance lze rozdělit do následujících kategorií [23]:

- *Pasivní obvodová redundance* – poskytuje odolnost proti poruchám pouze prostřednictvím maskování poruch, které vzniknou bez jakéhokoliv zásahu nebo opravy stavu systému.
- *Aktivní obvodová redundance* – vyžaduje zapojení mechanismu pro detekci poruchy, izolaci vlivu poruchy na funkci systému a obnovu stavu systému prostřednictvím odstranění komponenty s poruchou ze systému. Techniky založené na aktivní obvodové redundanci vyžadují, aby byl systém pozastaven a rekonfigurován.
- *Hybridní obvodová redundance* – kombinuje přístup pasivní a aktivní obvodové redundance. Maskování poruch je použito pro zamezení systému v generování chybných výstupů. Techniky detekce a lokalizace poruchy, spolu s obnovou stavu systému, jsou použity pro nahrazení poruchové komponenty rezervní komponentou. Hybridní techniky umožňují rekonfiguraci systému bez zastavení jeho provozu.

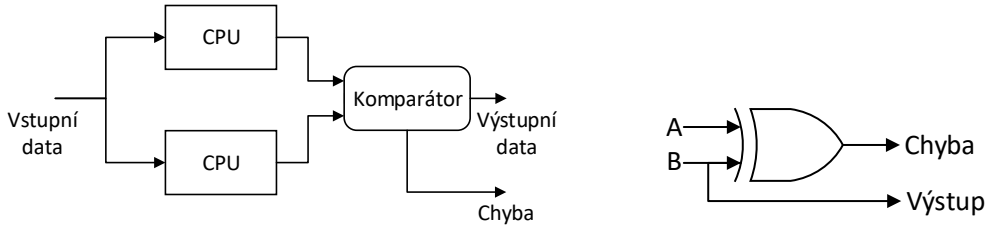
Nevýhodou použití obvodové redundance je obecně nárůst váhy, velikosti, spotřeby energie a ceny. Obvodová redundance může být aplikována na různých úrovních abstrakce systému. Redundance může být použita pro zálohování funkčních jednotek zařízení (tzn. na úrovni systému) nebo na úrovni elektronických obvodů. V případě obvodů FPGA se tato režie projevuje ve větší spotřebě zdrojů FPGA (programovatelných prvků, paměťových bloků, specializovaných funkčních bloků), nárůstem příkonu obvodu a v některých případech také omezením maximální pracovní frekvence obvodu [68].

Existuje mnoho různých způsobu využití redundance, které se snaží najít kompromis mezi přínosem obvodové redundance pro spolehlivost systému a zmíněnými nevýhodami, jenž jsou důsledkem redundance[23]. Využití TMR s sebou přináší nárůst spotřeby obvodových zdrojů o 200%. Pokročilými metodami pro implementaci TMR se bude zabývat blíže kapitola 5.

2.5.1 Duplexní systém s porovnáním

Duplexní a samo-kontrolující se systémy používají nejjednodušší formu obvodové redundance založené na zdvojení základního systému. Tento princip je označován jako *DMR (Dual Modular Redundancy)*. Duplexní systém s porovnáním je založen na principech aktivní obvodové redundance. Zabezpečená architektura disponuje komparátorem, který umožňuje

detekci poruch za pomoci porovnání výstupů dvou redundantních modulů. Za pomoci komparátoru lze signalizovat chybu na výstupech duplexního systému. Nicméně, komparátor již není schopný rozpoznat, ve kterém modulu systému vznikla porucha. Příklady implementace duplexního systému s porovnáním jsou znázorněny na obrázku 2.19.

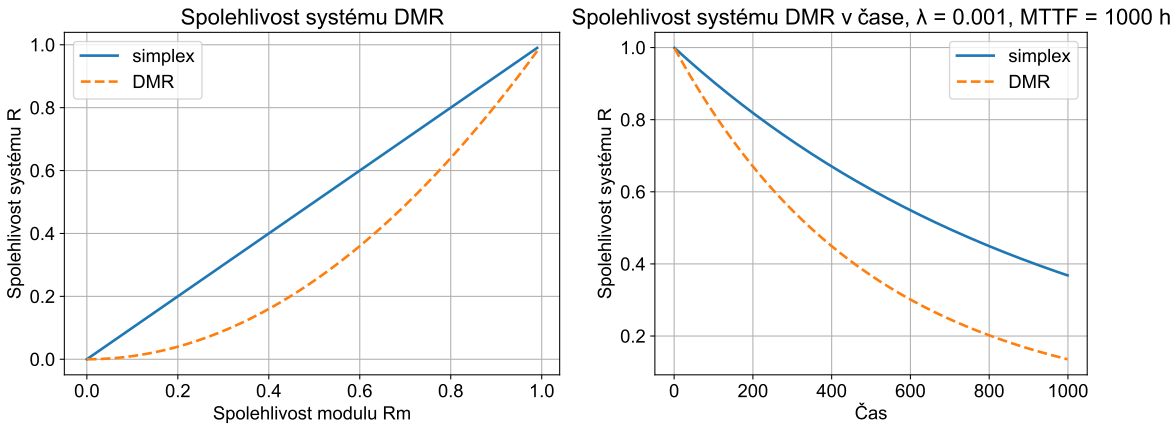


(a) Zdvojení procesorů (b) Implementace DMR pomocí log. hradel

Obrázek 2.19: Architektura duplexního systému s porovnáním výstupů

Spolehlivost R_{DC} duplexního systému s porovnáním, s předpokladem perfektního komparátoru (bezporuchového) a nezávislého selhání redundantních modulů, lze vyjádřit za pomoci vzorce 2.10. Obrázek 2.20a znázorňuje porovnání spolehlivosti duplexního systému s porovnáním R_{DC} a spolehlivosti nezabezpečeného systému R. Z porovnání spolehlivostí R_{DC} a R je názorné, že spolehlivost duplexního systému s porovnáním může být horší než u základního systému. Nicméně velkou výhodou duplexního systému zůstává schopnost detekce poruchy.

$$R_{DC} = R_1 * R_2 \tag{2.10}$$



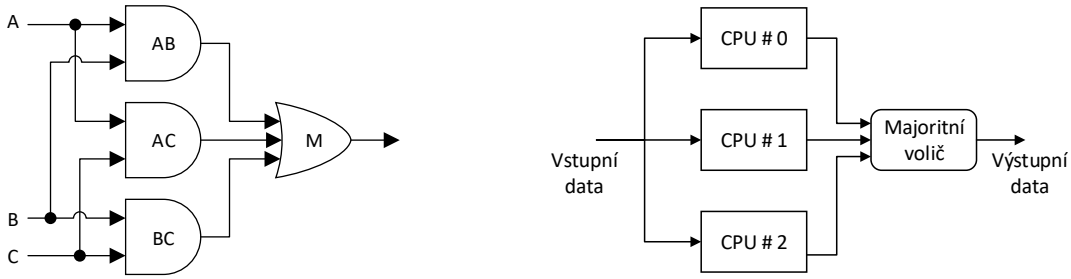
(a) Spolehlivost DMR systému (b) Spolehlivost DMR v čase

Obrázek 2.20: Porovnání spolehlivosti DMR a nezabezpečeného systému

Detekce poruchy je jedním z klíčových atributů bezpečnostně kritických systémů. Pro mnoho takových systémů není vyžadována odolnost proti poruchám, ale právě schopnost jejich detekce a možnost řízeného selhání a vypnutí funkce systému. Na základě reakce bezpečnostně kritického systému na poruchu se rozlišují systémy s bezpečným (předvídatelným) selháním (fail-safe), systémy s tichým selháním (fail-silent) a systémy pracující i při poruše (fail-operational).

2.5.2 Systém TMR

Trojitá modulární redundance (TMR) je nejpoužívanější technikou, jakou mohou být číslicové systémy zabezpečeny proti poruchám. Architektura TMR se skládá ze ztrojené konstrukce modulu paralelně provádějící stejný výpočet. Příklad zabezpečení procesoru pomocí TMR je znázorněn na obrázku 2.21b. Modul TMR představuje systém (nebo komponentu systému), který má být zabezpečen proti poruchám. Správný výstup TMR je zajištěn majoritním hlasováním na základě výběru správného výsledku z bezporuchových modulů. Implementace majoritního voliče pomocí tří logických hradel AND a jednoho logického hradla OR je zobrazena na obrázku 2.21a. Zapojení TMR umožňuje maskování poruchy jednoho redundantního modulu. Porucha dalšího ze zbývajících bezporuchových modulů by způsobila, že majoritní volič začne produkovat chybné výsledky.



(a) Implementace logiky majoritního voliče

(b) Procesor zabezpečený pomocí TMR

Obrázek 2.21: Architektura systému TMR

Spolehlivost TMR systému s perfektním majoritním voličem může být vyjádřena za pomoci vzorce 2.11. Spolehlivost TMR s exponenciálním rozložením pravděpodobnosti poruchy může být vypočtena dosazením $R(t) = e^{-\lambda t}$.

$$R_{TMR} = 3R^2 - 2R^3 \tag{2.11}$$

$$R_{TMR} = 3R^{2\lambda t} - 2R^{3\lambda t} \tag{2.12}$$

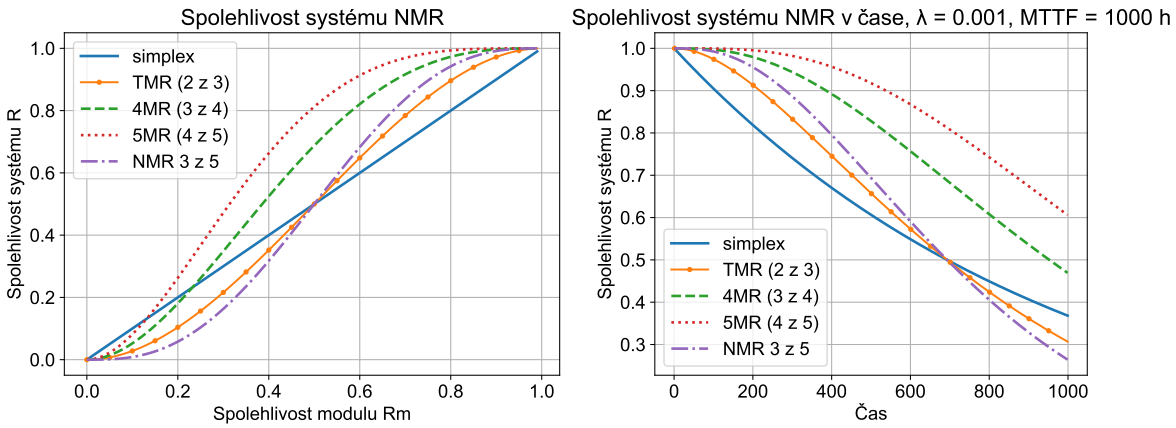
Předpokládáme-li, že majoritní volič není perfektní a má spolehlivost R_v, pak lze spolehlivost systému TMR vypočítat následovně:

$$R_{TMR} = (3R^2 - 2R^3)R_v \tag{2.13}$$

$$R_v = \frac{1}{3R - 2R} \tag{2.14}$$

2.6 Oprava poruch v obvodech SRAM FPGA pomocí rekonfigurace

Tato kapitola popisuje vybrané techniky pro opravu přechodných a trvalých chyb, které mohou být způsobeny nedestruktivními poruchovými jevy typu SEU (překlopení paměťového bitu) nebo typu SET (vznik přechodného pulzu signálu) v konfigurační vrstvě obvodu FPGA. Model poruch FPGA byl popsán v kapitole 2.3.5. Návrh systémů odolných proti



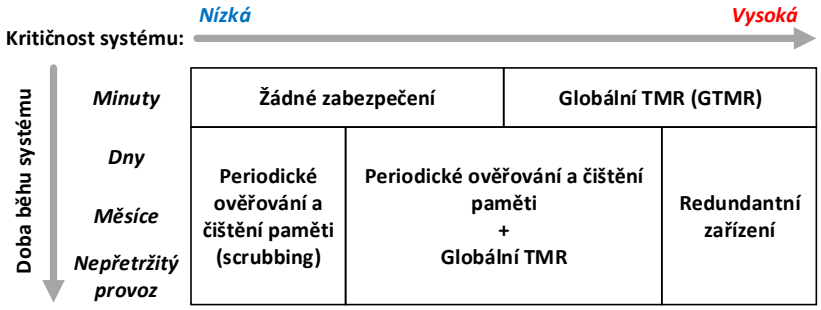
(a) Spolehlivost NMR systému

(b) Spolehlivost NMR v čase

Obrázek 2.22: Porovnání spolehlivosti TMR, NMR a nezabezpečeného systému

poruchám do obvodů FPGA se v praxi může řídit podle doporučení od firmy Xilinx [17], znázorněného na obrázku 2.23. Podle tohoto doporučení roste důležitost implementace techniky opravy stavu systému s požadavkem na vyšší bezporuchovost a dostupnost systému v závislosti na čase, i vzhledem k četnosti a typu uvažovaných poruch a kritičnosti systému.

Periodické čištění spolu s implementací globálního TMR je nejvhodnější technikou pro většinu případů použití. Tato metoda je schopná provést opravu přechodných a perzistentních poruch typu SEU v obvodech SRAM FPGA. Případně opravit poruchu typu MBU nebo větší počet akumulovaných poruch. Tato metoda je popsána v kapitole 2.6.1.



Obrázek 2.23: Metody pro zvýšení odolnosti proti poruchám používané v systémech implementovaných do FPGA v praxi

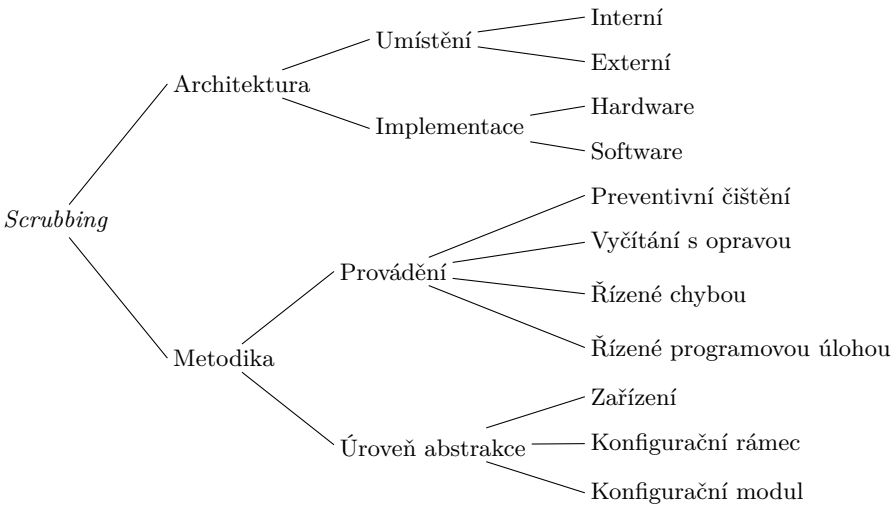
Alternativní metodou je využití částečné dynamické rekonfigurace pro opravu poruch v dedikovaných rekonfigurovatelných modulech systému zabezpečeného pomocí architektury TMR. Stejně jako v případě periodického čištění, částečná dynamická rekonfigurace slouží k odstranění přechodných a trvalých poruch v konfigurační paměti SRAM FPGA. Jedním z hlavních rozdílů je, že tento způsob rekonfigurace je schopen opravit poruchu ihned po její detekci. Tento způsob opravy poruch je vhodný pro systémy, které vyžadují větší flexibilitu a používají částečnou dynamickou rekonfiguraci také např. pro změnu a přizpůsobení své funkce. Metodika návrhu rekonfigurovatelných systémů odolných proti poruchám

je popsána v kapitole 5. Disertační práce na tuto metodiku dále navazuje a představuje vlastní metodiku pro návrh synchronizace stavu po dokončení opravy rekonfigurovatelného modulu, která je cílem této disertační práce.

Oprava stavu systému v případě vzniku permanentní poruchy v konfigurační paměti SRAM FPGA může být taktéž provedena pomocí částečné dynamické rekonfigurace. Nicméně, rozdílem oproti technikám periodického ověřování a čištění nebo rekonfigurace modulů TMR je, že je částečná dynamická rekonfigurace využita pro přemístění rekonfigurovatelného modulu do jiné vhodné části konfigurační paměti FPGA, která není ovlivněna permanentní poruchou. Tato metoda je popsána v podkapitole 2.6.2.

2.6.1 Periodické ověřování a čištění konfigurační paměti

Periodické ověřování a čištění konfigurační paměti, tzv. *scrubbing*, je technika pro odstranění poruch z konfigurační paměti FPGA založená na jejím periodickém kontrolování a přepisování chybných konfiguračních bitů. Obrázek 2.24 znázorňuje rozdělení technik pro implementaci metody *scrubbing* v FPGA. Řadič, který implementuje takovou techniku, se nazývá *scrubber*. Způsob implementace řadiče se odvíjí od vybrané obvodové platformy a způsobu jeho implementace. Mechanismus provádění opravy konfigurační paměti pak záleží na úrovni abstrakce adresování konfiguračních rámců v FPGA a vybrané metodě opravy.



Obrázek 2.24: Taxonomie technik periodického čištění konfigurační paměti

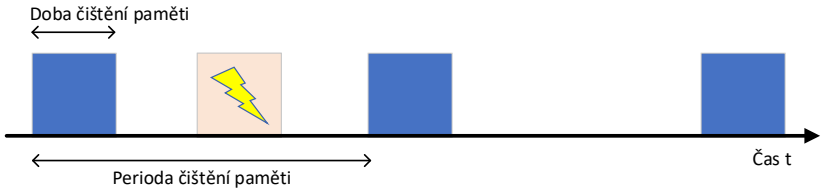
Samotný *scrubbing* není dostačující pro ochranu konfigurační paměti SRAM FPGA proti poruchám typu SEU. Důvodem je, že i při velmi rychlé frekvenci ověřování a opravy konfigurační paměti nelze zabránit nesprávnému chování programovatelné aplikační logiky v FPGA v době mezi vznikem poruchy a její opravou. Proto se scrubbing kombinuje s technikami založenými na obvodové redundanci, nejčastěji s TMR, které zaručí správnou funkci programovatelné logiky v době, než je konfigurační paměť opravena [103]. Nicméně, ani scrubbing ani implementace architektury TMR nezaručí odstranění poruch typu SEU v sekvenční logice implementované v aplikační vrstvě FPGA. Jednou z možností opravy stavu registrů je použíti techniky globálního TMR na úrovni RTL obvodů, kde všechny registry mají zavedeny zpětné vazby pro opravu svého stavu. Alternativně je nutné použít metodu založenou

na zpětné obnově, provést synchronizaci stavu v rámci systémové architektury TMR, nebo restartovat aplikační logiku a znovu inicializovat stav všech registrů.

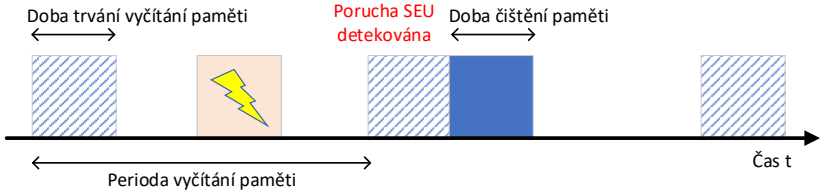
Principy návrhu řadiče pro ověřování konfigurační paměti

Návrh systému pro periodické ověřování konfigurační pamětí může být založen na několika principech [37]:

- *Způsob provedení opravného mechanismu* – Provádění ověřování konfigurační paměti může být prováděno preventivně nebo jako opravný mechanismus. Preventivní přepisování paměti bez jakékoliv logiky pro detekci chyb se označuje jako tzv. *blind scrubbing*. Obrázek 2.25 znázorňuje časování periodického čištění konfigurační paměti. Nejhorší případ může nastat při vzniku poruchy SEU ihned po dokončení cyklu čištění konfigurační paměti. Obrázek 2.26 znázorňuje použití pokročilejší metody, která kombinuje opravný scrubbing se zpětným vyčítáním obsahu paměti (angl. *read-back*). Opravný scrubbing a read-back může být rozšířen o výpočet kontrolního součtu CRC z dat vyčtených z konfigurační paměti FPGA.



Obrázek 2.25: Schéma časování periodického čištění konfigurační paměti

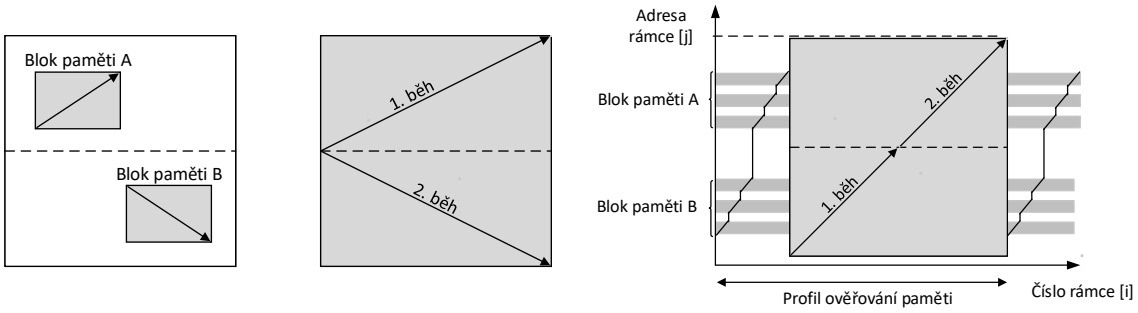


Obrázek 2.26: Schéma časování periodického čištění s vyčítáním konfigurační paměti

- *Frekvence ověřování konfigurační paměti* – Perioda preventivního čištění konfigurační paměti, tzv. *scrub rate*, nebo perioda opravného vyčítání, tzv. *readback rate*, bývají většinou nastaveny na pevnou hodnotu. Frekvence ověřování konfigurační paměti by měla být větší než je předpokládaná četnost poruch *SER* [37]. Podle standardu ECSS by měla být četnost ověřování alespoň 10-krát rychlejší než je nejhorší odhad četnosti poruch SER [27]. Nicméně taková implementace může mít někdy za následek návrh složitých implementací s velkými nároky na spotřebu energie. Alternativním řešením může být implementace řadiče, který umožňuje adaptivní změnu četnosti ověřování konfigurační paměti na základě okolních podmínek prostředí (např. výška družice na orbitě Země, předpokládaná četnost poruch SER).

- *Zpracování konfiguračních rámců* – Pokud je uvažována celá konfigurační paměť FPGA, pak se technika označuje jako tzv. *device-oriented scrubbing*. Zde lze využít konfigurační logiky v FPGA, které automaticky inkrementuje adresu FAR cílového konfiguračního rámce. Konfigurační rámce jsou přepisovány resp. vyčítány sekvenčně a adresa FAR je inkrementována automaticky. Jednoduchý scrubber může být konfigurován pro čištění celkové velikosti přepisované oblasti v konfigurační paměti. Druhou možností je návrh mechanismu čištění konfigurační paměti, který se stará o adresování konfiguračních rámců samostatně. Takový scrubber může provádět periodické vyčítání konfiguračních rámců a přepsat pouze ty rámce, v kterých byla detekována porucha.

- *Způsob adresování konfigurační paměti* – Konfigurační rámce v FPGA mohou být jednoduše procházeny sekvenčně v rámci 1D prostoru. Nicméně většinou není využita celá konfigurační paměť. Poruchy SEU vzniklé v nevyužité konfigurační paměti nebudou mít pravděpodobně žádný vliv, jestliže porucha nezasáhla propojovací logiku FPGA. Scrubbing může být prováděn tedy pouze pro část logiky, která je více citlivá na vznik poruch typu SEU. Taková logika může být fyzicky namapována do 2D oblasti a periodicky čištěna s vyšší frekvencí. Čištění konfiguračních rámců v rámci 1D a 2D prostoru může být kombinováno [14], kdy opravné vyčítání konfigurace může být prováděno pro definovaný modul v rámci 2D oblasti konfigurační paměti s vyšší četností a preventivní scrubbing může být prováděn s menší periodou v rámci celé konfigurační paměti.



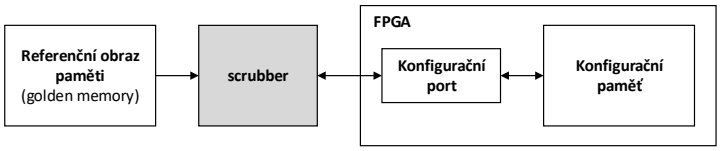
(a) Jednoduché adresování

(b) Kombinované 2D a 1D adresování

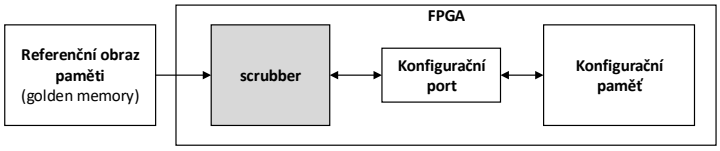
Obrázek 2.27: Způsoby adresování konfigurační paměti během periodického čištění

- *Výběr umístění pro implementaci řadiče pro ověřování paměti* – Scrubber může být implementován jako *interní* nebo *externí*[13]. Interní scrubber využívá rozhraní *ICAP* a je implementován ve stejném obvodu FPGA jako cílový systém, jehož konfigurace je ověřována. Externí scrubber může být implementován za pomoci mikrokontroléru, FPGA nebo ASIC obvodu. Externí scrubber přistupuje k ověřované paměti FPGA přes vnější konfigurační rozhraní (např. *SelectMAP*).

- *Výběr platformy pro implementaci řadiče pro ověřování paměti* – Scrubber může být implementován několika způsoby. V případě externího řadiče, může být celá logika implementována v odolném obvodě FPGA nebo ASIC. Alternativní možností je implementace v mikroprocesoru a návrh programově řízeného mechanismu periodické čištění.



(a) Externí scrubbing



(b) Interní scrubbing

Obrázek 2.28: Návrh externího a interního scrubbingu

2.6.2 Oprava systému z permanentní poruchy

Permanentní poruchy v konfigurační paměti obvodů SRAM FPGA mohou vzniknout především poškozením obvodu vlivem stáří nebo destruktivních důsledků působení kosmického záření. V takovém případě nelze opravit stav paměťových buněk konfigurační paměti, které mohou být trvale přivedeny na logickou úroveň 0 nebo 1. Jednoduchou metodou je využití aktivní obvodové redundance a deaktivaci nefunkčního modulu v rámci N-modulárního systému případně v rekonfiguraci zapojení N-modulárního systému a aktivaci záložního modulu [23]. Tato metodika může být použita na úrovni zařízení nebo obvodů.

U rekonfigurovatelných obvodů SRAM FPGA lze využít částečné dynamické rekonfigurace a provést rekonfiguraci FPGA do té podoby, že logika konfigurační bloku CLB s permanentní poruchou není využita [116] [15]. Pro opravu systému z permanentní poruchy lze využít dvě základní techniky:

- *Rekonfigurace FPGA s využitím předkompilovaných alternativních konfigurací* – Tato technika využívá předem vytvořené alternativní konfigurace FPGA, které jsou uloženy v nevolatilní paměti a využity jen v případě detekce permanentní poruchy. Během rekonfigurace je využita před-připravená konfigurace, která je nahrána do FPGA bez nutnosti opakování procesu propojování a mapování [116] [57] [58].

  Důležitým předpokladem využití této metody je, že konfigurace pro rekonfigurovatelné moduly PRM jsou vytvořeny za pomoci techniky relokace částečných konfiguračních souborů *PBR (Partial Bitstream Relocation)*.

  Příkladem může být metoda založená na rozdělení konfigurační paměti na rekonfigurovatelné dlaždice a přemístění konfigurace funkce dlaždice s permanentní poruchou do rezervních dlaždic [21]. Dlaždice definované v konfigurační paměti mohou plnit jednu ze tří různých funkcí: implementovat logiku provádějící výpočty a zpracování dat, být rezervní dlaždicí a nebo implementovat logické obvody pro propojení a vzájemnou komunikaci logických obvodů implementovaných ve výpočetní dlaždici. V případě detekce permanentní poruchy je logika nefunkční dlaždice přemístěna do některé z rezervních dlaždic. Propojovací dlaždice je rekonfigurována taktéž.

- *Rekonfigurace FPGA za pomoci konfigurace vytvořené za běhu* – Základní myšlenkou této techniky je opakovat procesy rozmístění a propojování za běhu systému, vygene-

rovat novou částečnou konfiguraci pro FPGA a dynamicky změnit obsah konfigurační paměti tak, aby nově využitá programovatelná logika nezahrnovala oblast konfigurační paměti s permanentní poruchou.

Příkladem může být obvodová jednotka navržená v [16], která je schopná provést algoritmus propojování logických prvků a přímo změnit konfiguraci FPGA za běhu. Systém odolný proti poruchám je implementován v obvodě FPGA Xilinx Zynq, který má vestavěný fyzický dvou-jádrový procesor ARM Cortex-A9. V případě poruchy procesor řídí proces opravy poruchy. Procesor předá informace o porušené cestě v FPGA jednotce provádějící dynamický návrh propojení, procesor následně vygeneruje novou konfiguraci pro částečnou dynamickou rekonfiguraci a přeprogramuje FPGA pomocí rozhraní ICAP.

2.7 Zotavení stavu systému z poruchy

Zotavení stavu systému z poruchy označuje přechod systému z provozního stavu systému, ve kterém se nacházejí chyby vzniklé vlivem působení přechodné nebo trvalé poruchy, do bezporuchového stavu, který chyby již neobsahuje. Pro odstranění přechodných a permanentních poruch z konfigurační paměti FPGA se využívají různé techniky založené na principech popsaných v kapitole 2.6 a schopnosti rekonfigurace FPGA.

Jestliže chceme udržet systém implementovaný v FPGA i nadále funkční, je nezbytné provést po opravě příčiny poruchy také opravu stavu systému a odstranit možné následky poruch v implementované aplikační logice.

Při opravě stavu systému jsou odstraněny chyby, které se mohly akumulovat v registrech sekvenční logiky např. vlivem působení poruchy SEU nebo SET. Pro opravu stavu registrů je nutné provést jejich opětovné přenastavení na správnou hodnotu.

V principu existují dva základní způsoby, jak obnovit stav systému do plné funkčnosti po vzniku poruchy. Vždy je nutné mít k dispozici data správného stavu systému.

- *Zpětná obnova stavu* – Stav systému může být obnoven zpětně na základě předem uloženého obrazu správného stavu systému (*roll-back recovery*). Obraz stavu systému je kopie všech proměnných důležitých pro aktuálně prováděné operace systému. Ukládání stavu systému probíhá průběžně během správné funkce systému v tzv. kontrolních bodech (*checkpoint*). Zpětná obnova systému vyžaduje, aby posloupnost kroků systému mezi kontrolním bodem a stavem, ve kterém byla detekována porucha, vedla vždy ke stejnému výsledku. Důvodem je nutnost opakování výpočtu systému od posledního uloženého stavu. Vzhledem k tomu, že opakování výpočtů i pravidelné ukládání dat stavu způsobuje časovou režii, je nutné zvážit poměr mezi trváním opakovaného výpočtu, četností ukládání dat stavu a také pravděpodobnosti výskytu poruchy systému. Zpětná obnova se typicky používá např. u duplexních systémů. Pro systémy reálného času je zpětná obnova stavu systému nevhodná, jelikož dochází k navrácení funkce systému do předchozího bodu. Vlivem časové režie a opakování výpočtu může dojít k nesplnění časových mezí stanovených pro funkci systému [116].

- *Dopředná obnova stavu* – Alternativou pro zpětné obnovení stavu systému jsou metody provádějící obnovu aktuálního stavu systému (*roll-forward recovery*). Tento přístup nevyžaduje pravidelné ukládání obrazu stavu systému ani opakování výpočtů. Metody dopředné obnovy stavu jsou často založené na využití obvodové, časové nebo informační redundance. V případě obvodové nebo programové redundance lze získat

správná data (obraz stavu systému) pro obnovu stavu modulu TMR z jiného funkčního modulu nebo redundantní výpočetní úlohy. V případě zabezpečení stavu systému pomocí informační redundance může být stav opraven pomocí korekčních kódů.

V ideálním případě by funkce systému odolného proti poruchám neměla být narušena. Systém využívající dopřednou obnovu stavu systému by měl po dokončení opravy stavu pokračovat ve své funkci dále.

V případě systému zabezpečeného pomocí architektury TMR lze pokládat všechny metody založené na dopředné obnově stavu za metody provádějící synchronizaci stavu opravovaného modulu TMR s ostatními redundantními moduly.

2.8 Formální definice problému synchronizace stavu

Základní princip využití architektury TMR pro zabezpečení číslicového systému byl popsán v kapitole 2.5.2. Hlavním důvodem pro aplikaci TMR v systémech odolných proti poruchám je schopnost maskování vlivu poruch při selhání jednoho redundantního modulu ze tří.

V minulosti bylo usuzováno, že přechodná porucha v systému zabezpečeném pomocí TMR po krátké době zmizí a nezanechá žádné následky. Předpokladem bylo, že díky TMR lze tolerovat jednorázové přechodné poruchy do té míry, dokud je v TMR ovlivněn pouze jeden modul. Tyto úvahy se týkaly jednoduchých číslicových obvodů, které využívali TMR na úrovni svých komponent. Nicméně při implementaci TMR na vyšších úrovních abstrakce je možné, že jednorázová přechodná porucha způsobí permanentní poruchu systému [101].

Jednoduchým příkladem může být implementace čítače zabezpečeného pomocí TMR, kdy se hodnota jednoho z redundantních čítačů změní vlivem výskytu poruchy typu SEU. Jestliže hodnota ovlivněného čítače není opravena, stává se vliv přechodné poruchy trvalým.

Tento problém se týká pouze sekvenčních obvodů zabezpečených pomocí TMR, jejichž stav musí být po odeznění vlivu poruchy nebo po dokončení opravy poruchy synchronizován s ostatními instancemi obvodu.

Definice 1 *Sekvenční obvod M může být reprezentován uspořádanou pěticí $\langle I, Q, Z, \delta, \omega \rangle$, kde I je množina všech vstupních vektorů, Q je množina stavů, Z je množina výstupních vektorů, $\delta : Q \times I \rightarrow Q$ je přechodová funkce pro určení následujícího stavu a $\omega : Q \rightarrow Z$ je výstupní funkce [100].*

Jestliže přechodná porucha ovlivní výpočet přechodové funkce δ, sekvenční obvod může provést přechod do chybného stavu q_e. Chybný stav q_e je různý od správného stavu q_c. Lze tedy usoudit, že sekvence vstupních vektorů i aplikována na stavy z q_e a q_c nebude vést opět ke stejnému stavu, tzn. $\delta(q_e, i) \neq \delta(q_c, i)$. Jinými slovy, jakmile je stav sekvenčního obvodu porušen, budoucí stavy, do kterých obvod přejde, mohou být taktéž nesprávné. Z dlouhodobého hlediska a v případě komplexnějších systému může mít porucha v sekvenčním obvodu jednoho redundantního modulu TMR vážné následky.

Definice 2 *Sekvenční obvod M je obnovitelný (tzv. restorable) jestliže platí, že pro každý pár stavů (q_e, q_c) existuje vstupní sekvence i_r pro kterou platí $\delta(q_e, i_r) = \delta(q_c, i_r)$. Taková vstupní sekvence i_r se pak nazývá obnovovací sekvencí (restoring sequence) [100].*

Definice 3 *Vstupní sekvence i_s je synchronizační sekvencí (tzv. synchronizing sequence) sekvenčního obvodu M jestliže platí, že pro každý stav q_j a nějaký fixní stav q_s existuje přechodová funkce $\delta(q_j, i_s) = q_s$ [100].*

Lze říci, že synchronizační sekvence je obnovovací sekvencí pro každou dvojici správných i chybných stavů, protože pro všechny (q_e, q_c) platí $\delta(q_e, i_s) = \delta(q_e, i_s) = q_s$. Dále lze usoudit, že pokud je synchronizační sekvence aplikována periodicky během normální funkce sekvenčního obvodu, vliv přechodných poruch může být eliminován.

Příkladem může být aritmeticko-logická jednotka hypotetického procesoru a změna hodnot registrů vlivem přechodné poruchy. Jednotka provádí aritmetické operace nad registry. Hodnota registru R0 může být pravidelně aktualizována na součet R0 a vstupních dat. Nicméně tato operace není schopna obnovit stav R0. Hodnota R0 může být obnovena pouze instrukcí `load`, která do registru uloží novou hodnotu bez jakékoliv závislosti na aktuálním stavu jednotky. Celá jednotka může být tedy synchronizována sekvencí instrukcí `load` provedenou pro všechny registry.

Pro některé sekvenční obvody nemusí existovat synchronizační sekvence. V praxi je výhodné implementovat synchronizační sekvenci v podobě dodatečného nulovacího vstupu, který umožní nejen inicializaci stavu obvodu při spuštění, ale také synchronizaci v případě výskytu přechodné poruchy.

Kapitola 3

Současný stav řešeného problému

Cílem této kapitoly je popsat a charakterizovat existující metody pro obnovu a synchronizaci stavu systému zabezpečeného pomocí tří-modulové obvodové redundance. Pro zajištění synchronizace stavu redundantních modulů architektury TMR lze využít různých způsobů založených na principech dopředné obnovy stavu. Existující metody pro synchronizaci stavu mohou být rozděleny z hlediska typu architektury TMR a úrovně abstrakce implementace TMR následovně:

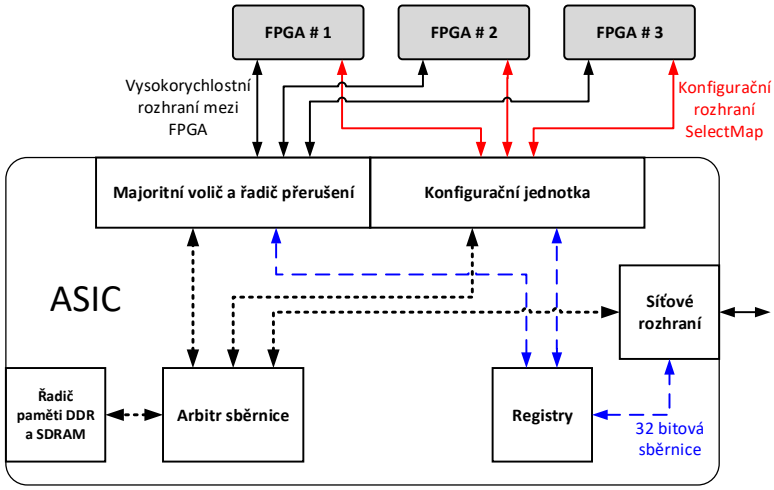
- synchronizace TMR na úrovni redundantních obvodů FPGA,
- synchronizace na úrovni víceprocesorového systému,
- synchronizace TMR na úrovni procesoru,
- synchronizace TMR na úrovni konečných automatů,
- synchronizace TMR na úrovni registrů sekvenční logiky.

3.1 Synchronizace TMR na úrovni redundantních obvodů FPGA

Fyzická replikace hardwarových jednotek nebo zdrojů je nejjednodušší formou implementace redundantního systému. Tento přístup využili taktéž autoři příspěvku [83], kteří navrhli rekonfigurovatelnou a poruchám odolnou platformu. Architektura navrženého systému je zobrazena na obrázku 3.1. Systém se skládá z obvodu ASIC odolného vůči radiaci a tří redundantních obvodů FPGA zapojených v architektuře TMR. Výstupní řídicí a datové signály obvodů FPGA jsou majoritně voleny uvnitř obvodu ASIC.

Hlavním předpokladem pro správnou funkci takového systému TMR je, že redundantní obvody FPGA musí operovat synchronně. Správný návrh propojení mezi obvody FPGA a obvodem ASIC musí zajistit, aby byla délka všech korespondujících si redundantních signálů stejná. Kromě řídicích a datových signálů je přenášen také hodinový signál, který zajišťuje správné vzorkování signálů. Řadič konfigurace *Configuration Manager* disponuje třemi nezávislými konfiguračními rozhraními *SelectMAP*, což umožňuje nezávislou konfiguraci každého z obvodů FPGA.

Autoři [83] navrhli také několik technik pro synchronizaci stavu redundantních obvodů FPGA. Synchronizace stavu obvodů FPGA následuje po rekonfiguraci jednoho obvodu FPGA, v němž byla detekována porucha. Synchronizace pro systém implementující architekturu znázorněnou na obrázku 3.1 může být realizována několika způsoby:



Obrázek 3.1: Synchronizace stavu redundantních obvodů FPGA [83]

a) *Současný restart obvodů FPGA* – Nejjednodušším způsobem synchronizace je současný restart všech obvodů FPGA. Restart lze provést pouze tehdy, pokud se systém opakovaně dostává do vhodného a bezpečného stavu.

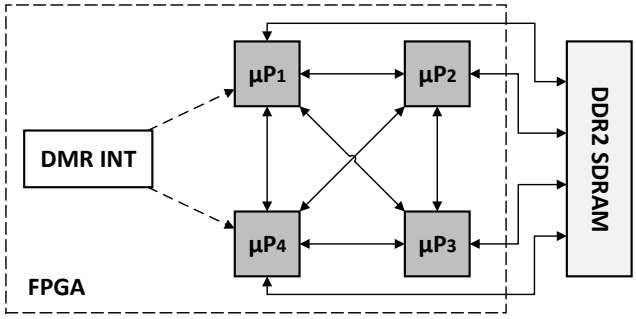
b) *Synchronizace obvodů FPGA po dokončení zpracování* – Dalším uvažovaným scénářem je situace, kdy systém opakovaně zpracovává bloky dat. Příkladem mohou být typicky komunikační systémy a aplikace zpracovávající obrazová či radarová data. Zpracování dat může probíhat na pozadí a po dokončení prováděné úlohy může být systém restartován.

c) *Obnova stavu obvodu FPGA ze zálohy* – Stav komplexnějších systémů, provádějících kritické výpočty s využitím adaptivních řídicích algoritmů a hodnot vypočtených v předcházejících krocích, musí být nejdříve zálohován a připraven pro obnovu stavu porušeného obvodu FPGA. Následně všechny tři obvody FPGA mohou být rekonfigurovány a jejich stav obnoven. Případně pouze nefunkční obvod bude rekonfigurován a všechny tři obvody synchronizovány ve vhodném okamžiku, v němž je stav poškozeného obvodu FPGA obnoven ze zálohy.

3.2 Synchronizace na úrovni víceprocesorového systému

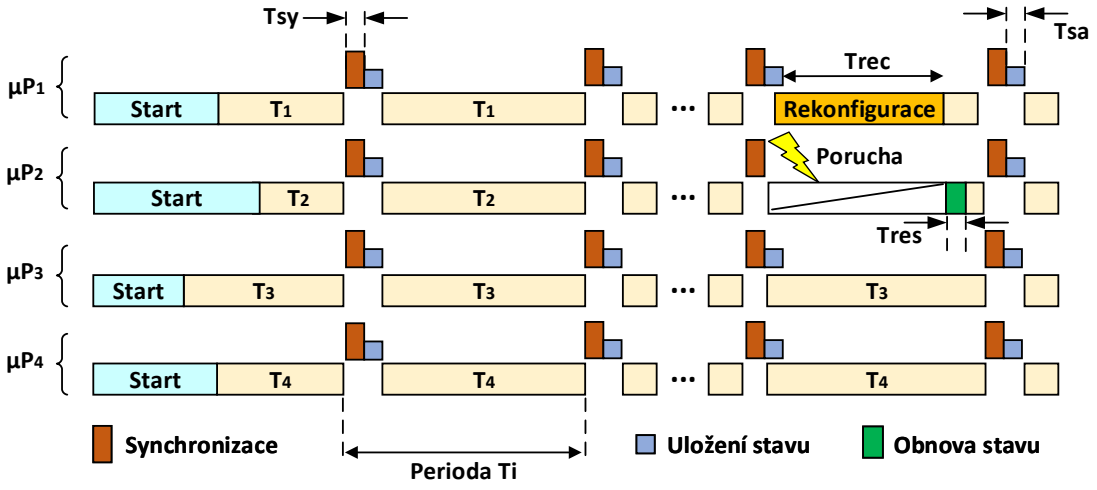
Multi-procesorový systém na čipu *MPSoC (Multi-Processor System on a Chip)* je tvořen několika vzájemně propojenými procesorovými jádry, které jsou integrovány na jednom čipu. Procesorová jádra spolu s pamětí a obvody periferních zařízení jsou většinou uloženy do samostatných funkčních jednotek. Tyto funkční jednotky mohou být vzájemně propojeny pomocí komunikační sběrnice nebo integrované sítě na čipu *NoC (Network on Chip)*. Funkční jednotky mohou také komunikovat prostřednictvím sdílené paměti. Každá funkční jednotka je schopná provádět celou nebo část úlohy. Systémy MPSoC mohou být implementovány za pomoci programovatelné logiky obvodu FPGA. Funkční jednotka může být v případě výskytu přechodné nebo permanentní poruchy rekonfigurována.

Na obrázku 3.2 je znázorněna rekonfigurovatelná MPSoC architektura navržená autory článků [75] [74] [76]. Navržená architektura systému se skládá ze čtyř procesorových



Obrázek 3.2: Architektura víceprocesorového systému MPSoC [74]

jednotek, které vzájemně komunikují prostřednictvím sběrnice *FSL* (Fast Simplex Link). Každá procesorová jednotka se skládá z soft-core procesoru *Xilinx MicroBlaze* a periferií. Procesory sdílí přístup k řadiči DDR2 paměti SDRAM, konfiguračním rozhraním ICAP a sběrnici FSL. Tyto komponenty jsou implementovány ve statické části návrhu. Každý z procesorů má dedikované rozhraní, implementované pomocí tzv. *bus makra*, které poskytuje jednotný přístup k procesorům a umožňuje odpojit procesor od zbytku systému v případě jeho poruchy či rekonfigurace.



Obrázek 3.3: Časový diagram rekonfigurace a synchronizace ve více-procesorovém systému

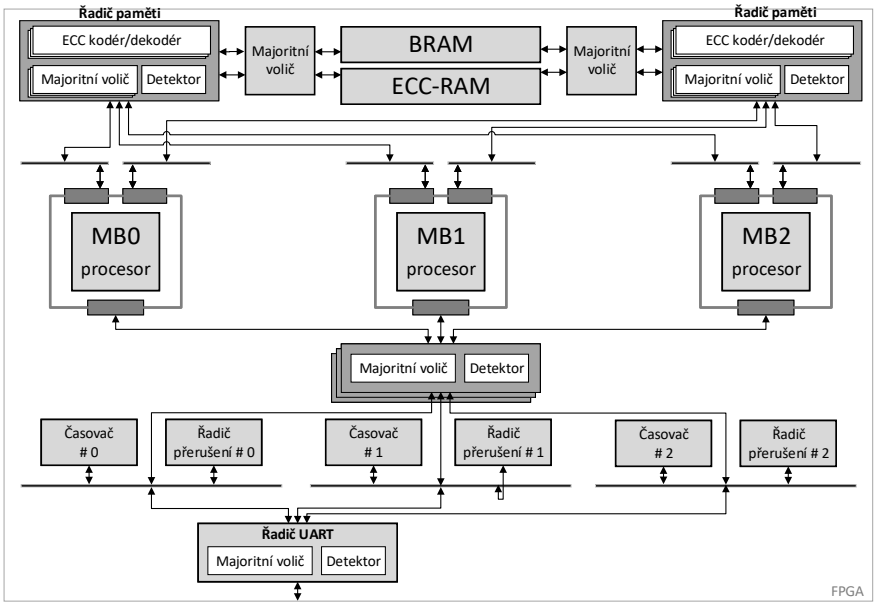
Rekonfigurace procesorové jednotky umožňuje změnit naprogramovanou úlohu procesoru, použitá periferní zařízení nebo celou obvodovou logiku jednotky. Kompletní rekonfigurace procesorové jednotky je provedena v případě detekce poruchy nebo pokud má jiná jednotka převzít prováděnou úlohu. Všechny procesory jsou periodicky synchronizovány pomocí externího přerušení, což umožňuje provést krátkou programovou proceduru nezávislou na prováděné úloze v procesorech. Časový diagram provozu jednotlivých procesorových jader je znázorněn na obrázku 3.3. Na začátku synchronizace všechny procesory vstoupí do obslužné rutiny přerušení. Každý procesor během této doby uloží svůj stav do externí paměti DDR2 a provede výměnu synchronizačních rámců s ostatními. Synchronizace neslouží

pouze k obnově stavu procesorů ale také k detekci funkčnosti vzájemného propojení. Pokud některý z procesorů nepřijme synchronizační data od některého z procesorů během definované doby, znamená to poruchu propojení. Pokud všechna spojení mezi daným procesorem a ostatními selžou, znamená to selhání celého procesoru. Následně se funkční procesory dohodnou, který z nich provede rekonfiguraci nefunkčního uzlu.

3.3 Synchronizace TMR na úrovni procesoru

Procesor může být do návrhu v FPGA snadno integrován ve formě tzv. měkkého jádra (*soft-core*) procesoru. U typického procesoru musí být synchronizován programový čítač, ukazatel zásobníku, obsah zásobníku, stavový a řídicí registr, programové registry určené pro všeobecné použití a vnitřní programová paměť.

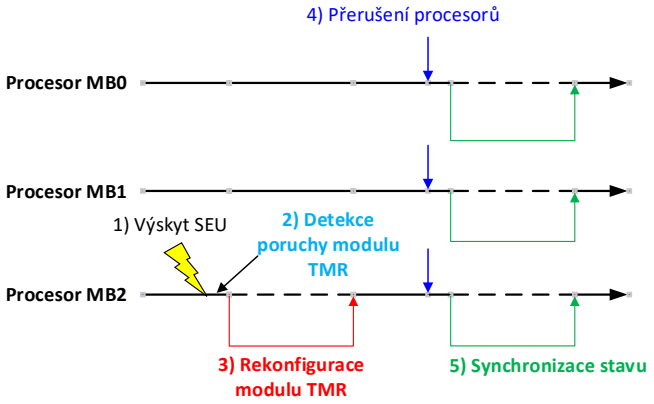
Autoři série článků [96] [43] [42] popsali techniku pro synchronizaci stavu procesoru *Xilinx MicroBlaze* zabezpečeného pomocí architektury TMR se sdílenou pamětí BRAM zabezpečené pomocí ECC. Navržená architektura je znázorněna na obrázku 3.4. Každé z procesorových jader je umístěno v určeném rekonfigurovatelném regionu FPGA, který může být v případě detekce poruchy rekonfigurován. Paměť BRAM je implementována jako dvou-portová, což umožňuje oddělený přístup do programové a datové části paměti. ECC využívá Hammingova kódu pro zabezpečení dat. Logika ECC je schopna opravit jednu poruchu a detekovat dvě poruchy v jednom zabezpečeném bitovém slově. Periferní zařízení procesorů jsou přístupná přes sběrnici *OPB (On-chip Peripheral Bus)*, které ale zůstávají nechráněny. Výstupy procesorů a výstupy z periferních zařízení (čítače, časovače) jsou majoritně voleny. Majoritní volič je rozšířen o logiku pro detekci poruchy.



Obrázek 3.4: Návrh synchronizace procesoru MicroBlaze se sdílenou pamětí BRAM [43]

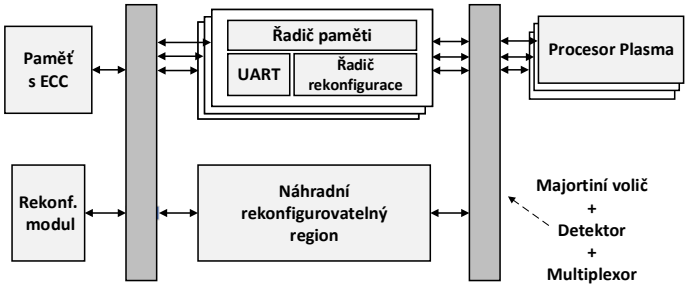
V případě detekce poruchy v jednom z modulů TMR je příslušná rekonfigurovatelná oblast rekonfigurována. Po dokončení rekonfigurace je pomocí externího přerušení spuš-

těna procedura synchronizace. Časový diagram celého procesu opravy poruchy systému je znázorněn na obrázku 3.5.



Obrázek 3.5: Časový diagram rekonfigurace a synchronizace procesoru MicroBlaze [43]

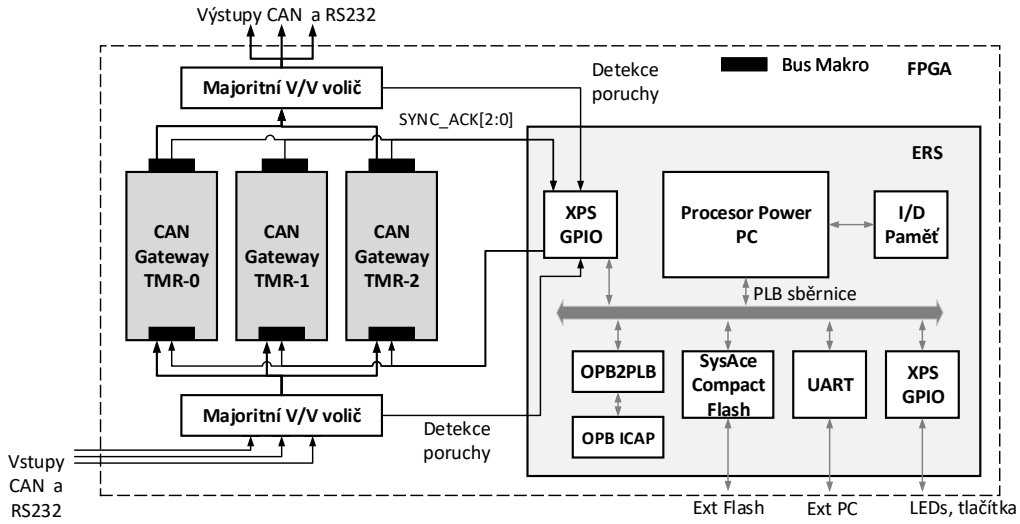
Na práci autorů [96] [43] [42] navázal příspěvek [32], který prezentuje systém odolný proti poruchám založený na 32-bitovém mikroprocesoru *Plasma* zabezpečeném pomocí architektury TMR. Redundantní procesorová jádra sdílí taktéž přístup k paměti BRAM, která je chráněna pomocí ECC. Architektura systému je zobrazena na obrázku 3.6. Návrh systému umožňuje obnovu z přechodných i trvalých poruch v obvodu SRAM FPGA. Pro opravu přechodných poruch v konfigurační paměti je použito periodické čištění. Při opravě systému z trvalé poruchy je nefunkční procesorové jádro rekonfigurováno a přesunuto do náhradního rekonfigurovatelného modulu. Synchronizace procesorů je provedena uložením stavu procesorů do paměti BRAM a zpětným vyčtením. Pro obnovu stavu registrů v periferních zařízením je použita technika implementace TMR se zpětnou vazbou pro opravu hodnoty registru.



Obrázek 3.6: Návrh synchronizace procesoru Plasma pomocí sdílené paměti BRAM [32]

Autoři [6] [66] [67] představili autonomní systém odolný proti poruchám. Architektura autonomního systému odolného proti poruchám je znázorněna na obrázku 3.7. Systém implementuje rozhraní mezi sběrnicí CAN a asynchronním komunikačním rozhraním *UART*. Jádrem systému je architektura TMR se třemi mikroprocesory *PicoBlaze* označenými jako *CAN GATEWAY TMRn*. Každý mikroprocesor obsahuje také řadič rozhraní *UART* a řadič sběrnice *CAN* s rozhraním *Wishbone*. Výstupy procesorů jsou majoritně voleny. *Voter* je také schopen indikovat modul TMR, v kterém byla detekována porucha. Oprava poruch

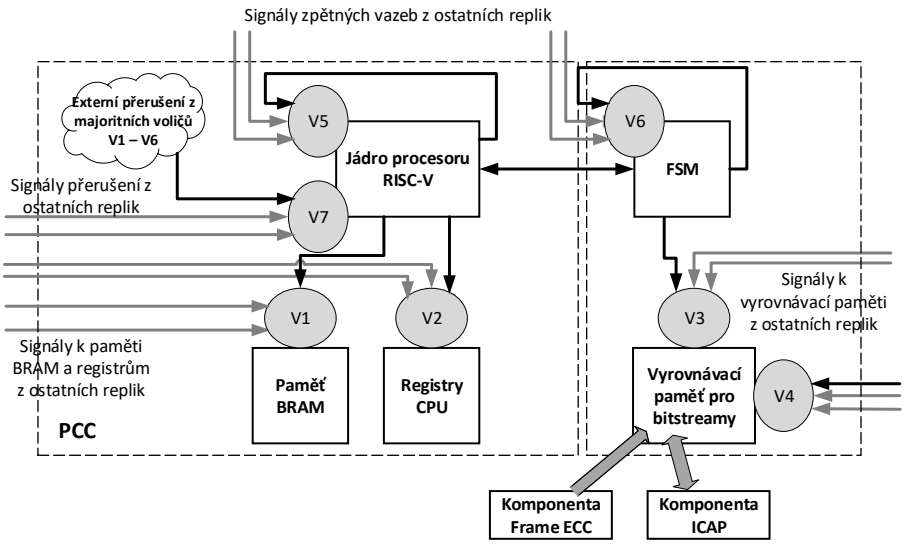
je zajištěna komponentou *ERS (Embedded Reconfiguration System)*, která umí provádět periodické čištění konfigurační paměti i dynamickou rekonfiguraci určeného modulu TMR. Mikroprocesor *Picoblaze* je modifikován pro podporu synchronizace stavu. Synchronizace procesorů je provedena v obsluze přerušení, kde neporušené procesory vystaví svá vnitřní data na sdílenou sběrnici *Wishbone* a rekonfigurovaný procesor je použije ke své synchronizaci. Výstupy procesorů jsou během synchronizace přivedeny do bezpečného stavu a nastaveny na neaktivní hodnotu. Díky tomu mohou okolní obvody detekovat, že je proces synchronizace aktivní.



Obrázek 3.7: Návrh synchronizace procesoru PicoBlaze pomocí sběrnice Wishbone s přivedením výstupů do bezpečného stavu [6]

Autoři [34] [35] navrhli metodu synchronizace stavu procesoru *ASIP (Application specific Instruction Set Processor)* s aplikačně specifickou instrukční sadou založeného na architektuře *RISC-V*, navržený speciálně pro potřeby řadiče rekonfigurace zvaného *PCC (Programmable Configuration Controller)*. Architektura PCC je znázorněna na obrázku 3.8. TMR je v systému aplikováno selektivně na několika místech pro zabezpečení vnitřních funkčních jednotek. Funkční jednotky uvnitř replikovaných procesorů jsou vzájemně propojeny. Majoritní voliče jsou vloženy na vstupy do paměti BRAM, souboru registrů, vyrovnávací paměti a FIFO pro uchování dat konfiguračního bitstreamu. Synchronizující majoritní voliče se zpětnou vazbou jsou vloženy pro opravu stavu programového čítače a registrů vnitřního konečného automatu.

Porucha v PCC detekovaná pomocí některého z majoritních voličů vygeneruje požadavek na přerušení procesoru. PCC poté pozastaví prováděnou aplikační úlohu (např. čištění konfigurační paměti). Procesor v obsluze přerušení provede uložení stavu svých registrů do blokové paměti BRAM pro jejich pozdější obnovu. Následně je přenesen konfigurační bitstream PCC z externí flash paměti odolné proti záření do konfiguračního rozhraní ICAP a provedena rekonfigurace poškozené repliky PCC. Synchronizace stavu repliky PCC s ostatními replikami se provede obnovou zálohy registrů z paměti BRAM. Proces obnovy stavu repliky PCC končí návratem z ISR.



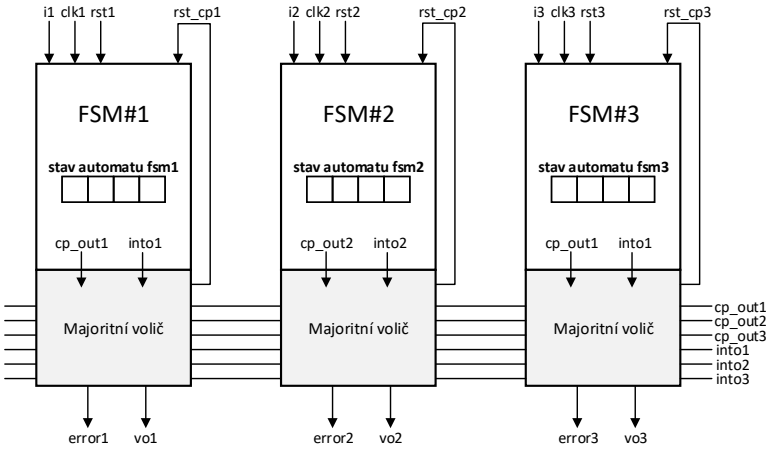
Obrázek 3.8: Návrh synchronizace procesoru ASIP pomocí vzájemně propojených redundantních pamětí BRAM a registrů se zpětnými vazbami

3.4 Synchronizace TMR na úrovni konečných automatů

Obvody implementované v FPGA jsou často řízeny konečnými automaty. Jednoduchým způsobem synchronizace stavu systému TMR řízeného konečným automatem je celkové restartování běhu modulů TMR nebo přivedení konečných automatů do stejného stavu. Toto řešení může být vhodné zejména pro periodické zpracování dat (např. paketů síťové komunikace nebo dat z radaru).

Autoři článků [77] [8] [9] představili techniku pro synchronizaci stavu v rekonfigurovatelném systému odolném proti poruchám řízeným konečným automatem *FSM* (Finite State Machine). Autoři se zaměřili na případ, kdy je odolný systém neustále v provozu. Synchronizace systému je prováděná ve stavech řídicího konečného automatu, které se během funkce systému pravidelně opakují. I když lze teoreticky každý stav konečného automatu použit k synchronizaci, v praxi se na stavy systému vztahují specifická omezení a tudíž ne všechny stavy jsou vhodné k provedení synchronizace. Pro minimalizaci doby k resynchronizaci systému je vhodné, aby se vybraný stav systému často opakoval. Pokud se na přechody mezi stavy vztahují nějaké podmínky, je nutné zajistit jejich splnění. Počet synchronizačních stavů ovlivňuje šířku synchronizační sběrnice nebo signálu, který propojuje redundantní moduly nebo majoritní volič.

Hlavní myšlenkou navržené metody synchronizace je použití speciálního obvodu *PRED* pro předvídání stavu, do kterého konečný automat následně přejde. Daný obvod obsahuje speciální vyhledávací tabulku s kombinacemi možných stavů a vstupů obvodu. Díky této tabulce lze určit za pomoci znalostí hodnot aktuálních vstupů a stavu systému příští synchronizační stav. Navržené řešení je znázorněno na obrázku 3.9. Řídicí konečný automat byl upraven, aby mohl aktivovat synchronizační stav pomocí signálu *cp*. Signál *rst_cp* umožňuje pozdržet nebo opět spustit konečný automat. Předvídací modul *PRED* implementuje dvě funkce:



Obrázek 3.9: Synchronizace konečných automatů s předvídáním stavu [8]

1. Předvídá budoucí synchronizační stav FSM, jenž je dále indikován signálem *cp* z majoritně volených výstupů. (Vybraných stavů určených k synchronizaci může být v automatu více.)

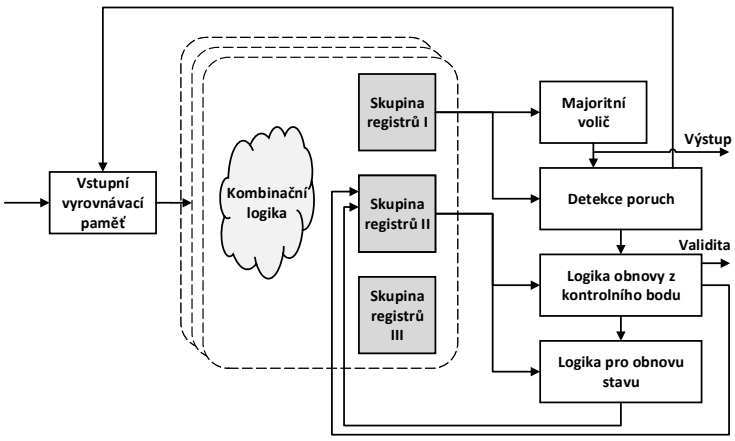
2. Řídí signál *rst_cp*, který spouští běh FSM synchronizovaného se zbytkem systému. Jakmile funkční modul TMR přejde do cílového stavu, konečný automat v rekonfigurovaném TMR modulu je uvolněn a spolu s ostatními automaty pokračuje v další činnosti.

3.5 Synchronizace TMR na úrovni registrů sekvenční logiky

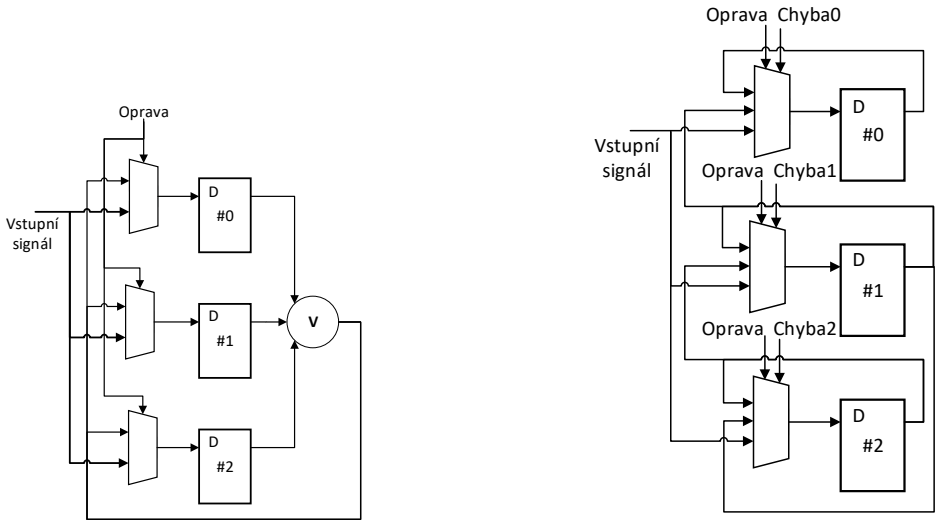
Synchronizace stavu systému a oprava hodnot registrů v případě výskytu poruchy typu SEU v aplikační logice implementované v FPGA může být zajištěna přímo na úrovni RTL.

Autoři [115] navrhli dvě schémata pro obnovu hodnot registrů uvnitř architektury TMR. Navržená odolná architektura se skládá ze tří obvodových kopií systému, vstupního vyrovnávacího registru, majoritního voliče připojeného na výstupy systému, detektoru poruch a logiky pro řízení kontrolních bodů a obnovy stavu systému. Schéma architektury je zobrazeno na obrázku 3.10. Tato technika pracuje se třemi různými skupinami registrů a proto vyžaduje detailní znalost o implementaci systému. Registry, které obsahují výstupní data, jsou přístupné přes majoritní volič. Registry, které obsahují data potřebná pro další funkci obvodů, jsou zabezpečeny pomocí opravné logiky. Registry, které obsahují dočasné mezivýsledky, není nutné zabezpečit. Implementovaná funkce systému se během bezporuchového běhu nemění a výstupy redundantních modulů systému jsou určeny pomocí majoritních voličů. Případná porucha v jakémkoliv z modulů je zaznamenána detektorem poruch na základě majoritního porovnání výstupů.

Na obrázcích 3.11a a 3.11b jsou znázorněna dvě schémata pro zapojení opravné logiky. První je volící schéma, které vybírá správnou hodnotu majoritním volením výstupů. Volená hodnota je zapsána do všech tří kopií registrů pouze v případě detekované poruchy. To je zaručeno implementací dodatečného multiplexoru před každý registr. Při bezporuchovém stavu jsou použity původní vstupy registrů. Druhé opravné schéma umožňuje přímé přepsání chybných dat v registrech vložením hodnot z přilehlých registrů s platnými daty.



Obrázek 3.10: Architektura pro obnovu stavu registrů na úrovni RTL pomocí volícího a přímo přepisujícího schématu [115]

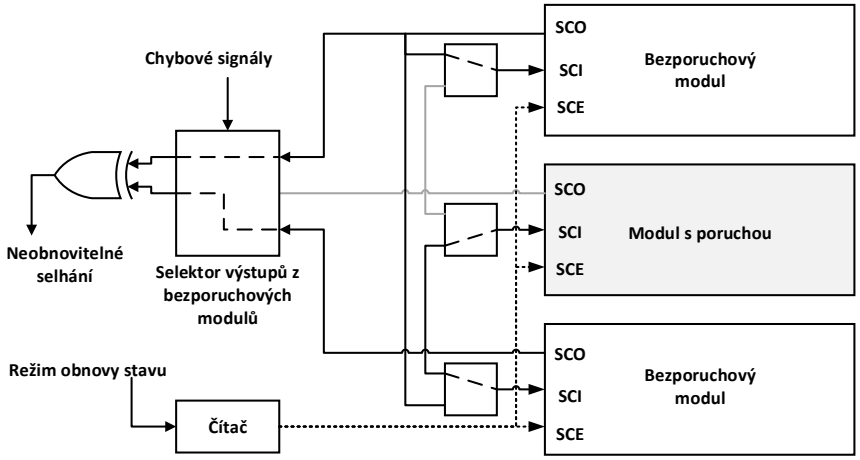


(a) (b)

Obrázek 3.11: Opravná logika volícího a přímo přepisujícího schématu pro obnovu stavu registrů [115]

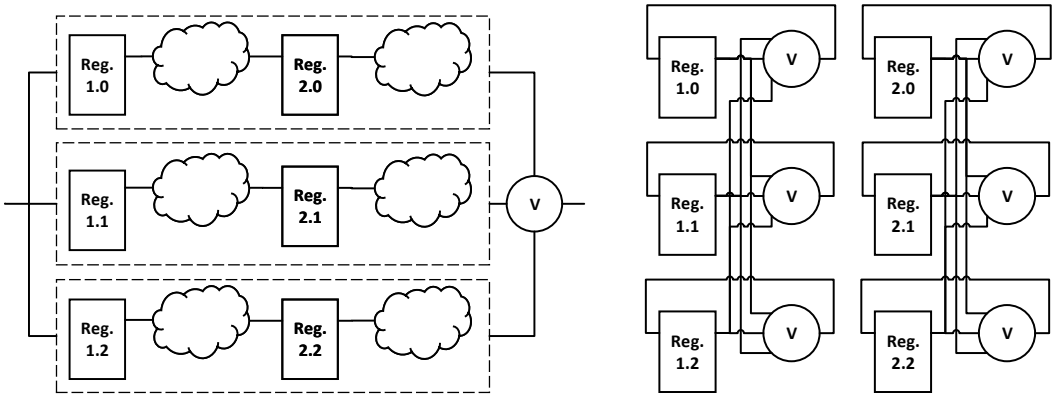
Další technikou využívající propojení vnitřních registrů byla publikována v [24]. Metoda nazvaná *ScTMR (Scan-chained TMR)* je založena na zřetězeném propojení registrů a vytvoření pomyslného posuvného registru, který má jeden sériový datový vstup a jeden sériový datový výstup. Tento princip se využívá u metod úplný a částečný *scan*, které se využívají pro zvýšení testovatelnosti návrhu číslicového systému. Schéma zapojení architektury ScTMR je znázorněno na obrázku 3.12. Zřetězené registry pracují jako jeden dlouhý posuvný registr. Pro snížení časové náročnosti autoři navrhují použít více paralelních řetězců. Odolná architektura se skládá ze tří redundantních modulů, majoritního voliče a řadiče ScTMR. Řadič ScTMR provádí kopii stavu registrů bezporuchového modulu do re-

gistrů modulu, ve kterém byla detekována porucha. V případě detekce poruchy se aktivuje opravná logika pomocí signálu *SCE* a pomocí multiplexeru se zvolí, jak budou moduly vzájemně propojeny. Sériový výstup *SCO* bezporuchového modulu je přiveden na sériový vstup *SCI* modulu, ve kterém byla detekována porucha a současně opět na vstup bezporuchového modulu. Tímto způsobem se zajistí, že po dokončení operace scan jsou registry v opravovaném modulu obnoveny na správnou hodnotu a hodnoty registrů ve zdrojovém modulu neporušeny. Autoři dále svůj výzkum rozšířili v [25] o možnost provádět souběžnou detekci poruch a tedy detekovat i poruchy, které by původní návrh ScTMR neodhalil. Novou architekturu nazvali *SMERTMR (Scan-chain-based Multiple Error Recovery TMR)*.



Obrázek 3.12: Schéma opravné logiky ScTMR [24]

Autoři příspěvku [81] představili metodu pro implementaci obnovy stavu registrů v architektuře TMR, která využívá rekonfigurace modulu pro odstranění poruchy a opravnou logiku propojující ztrojené registry napříč rekonfigurovatelnými moduly TMR. Navržená technika využívá nástroj, který umožňuje vytvořit dvě verze výsledné implementace logického návrhu systému. První verze odpovídá návrhu systému pro normální provoz. Implementovaný systém je v této verzi zabezpečen pouze pomocí jeho triplikace v rámci architektury TMR. TMR slouží k maskování a detekci přechodných poruch v konfigurační paměti. Druhá verze implementace slouží pro opravu stavu registrů na RTL úrovni. Schémata provozní a opravné logiky implementace jsou znázorněny na obrázku 3.13. Základním principem této techniky je, že při generování opravné verze implementace je využita znalost o pozicích všech vnitřních registrů automaticky extrahovaná při implementaci provozní verze návrhu. Díky tomu může být stav registrů v rámci opravné implementace uchován nezměněn. Opravná implementace návrhu přidává k registrům dodatečnou logiku sloužící k jejich vzájemné synchronizaci s ostatními instancemi nacházejícími se v ostatních modulech TMR. Pro synchronizaci všech registrů je nutné spustit opravnou verzi návrhu alespoň na jeden hodinový cyklus. Po provedení synchronizace stavu registrů za pomoci opravného návrhu je systém opět rekonfigurován za pomoci konfigurace pro provozní zabezpečený návrh systému.



(a) Zabezpečený návrh obvodu bez opravy (b) Opravný návrh obvodu se zpětnými vazbami

Obrázek 3.13: Schéma zabezpečeného a opravného návrhu pro obnovu stavu systému TMR na úrovni RTL [81]

3.6 Shrnutí

Na základě popisu a zhodnocení existujících metod pro synchronizaci stavu redundantních modulů TMR byla sestavena tabulka 3.1. Tabulka znázorňuje vhodnost použití různých principů metod synchronizace při různých úrovních abstrakce použití architektury TMR.

Metoda synchronizace	FPGA	MPSoC	CPU	FSM	RTL
Reset	✓	✓	✓	✓	✓
Synchronizace pomocí předvídání stavu				✓	
Synchronizace pomocí komunikační sběrnice	✓	✓	✓		
Synchronizace pomocí sdílené paměti na čipu		✓	✓		
Synchronizace pomocí HW propojení registrů			✓	✓	✓
Automatické vložení synchronizačních voličů					✓

Tabulka 3.1: Vyhodnocení vhodnosti metod pro synchronizaci v TMR pro různé obvodové platformy

Kapitola 4

Cíle disertační práce

Cílem této práce je *vytvoření metodiky pro návrh a implementaci číslicových obvodů pro synchronizaci stavu rekonfigurovatelných modulů TMR architektury systému odolného proti poruchám implementovaného do FPGA*. Metoda synchronizace navržená podle této metodiky by měla splňovat předem stanovená kritéria pro synchronizaci a zotavení stavu systému po výskytu a opravě poruchy.

4.1 Návaznost disertační práce na již existující metodiky

Vytvořená metodika je postavená na využití tří konceptů: implementace architektury TMR pro zabezpečení systému proti poruchám, schopnosti PDR moderních obvodů FPGA umožňující odstranění poruchy a zotavení stavu systému z poruchy. Tyto koncepty byly popsány v kapitolách 2.4, 2.6 a 2.7. Principy pro zabezpečení systému pomocí různých variant architektur TMR a technika návrhu rekonfigurovatelného systému byly blíže popsány v kapitole 5. Za způsob zotavení stavu systému z poruchy lze v případě architektury TMR pokládat synchronizaci stavu mezi redundantními moduly.

Existující metody pro synchronizaci stavu, které byly navrženy pro různé architektury systémů a aplikované na několika úrovních abstrakce návrhu architektury TMR, byly popsány v kapitole 3.

Mezi alternativní metody pro synchronizaci stavu TMR lze zařadit i využití synchronizujících majoritních voličů pracujících na úrovni FG-TMR, jež byly popsány v kapitole 5.1.2. V praxi je FG-TMR často kombinováno s periodickým čištěním konfigurační paměti.

4.2 Stanovení cílů

Pro vytvoření dané metodiky se předpokládá splnění následujících dílčích cílů:

1. Prozkoumat možnosti návrhu rekonfigurovatelného systému na platformě FPGA. Vytvořit rekonfigurovatelný návrh systému, pro který bude možné navrhnout specifické metody synchronizace stavu.

2. Vytvoření metodiky pro návrh metod synchronizace stavu. Metodika by se měla zaměřit na synchronizaci stavu v systémech řízených stavovými automaty a systémech řízených jádrem procesoru. Součástí metodiky by mělo být také stanovení základních kritérií pro návrh metod synchronizace, které umožní tyto metody parametrizovat, vyhodnocovat a optimalizovat.

3. Návrh a experimentální ověření mechanismů synchronizace stavu rekonfigurovatelných modulů pro systém řízený konečnými automaty.

4. Návrh a experimentální ověření mechanismů synchronizace stavu rekonfigurovatelných modulů pro systém řízený procesorem.

5. Návrh digitálních obvodů realizujících synchronizaci stavu rekonfigurovatelných modulů TMR systému jako odolných proti poruchám.

6. Navrhnout a implementovat verifikační prostředí pro ověření správné funkce rekonfigurace vybraného modulu v rámci architektury TMR a správného provedení synchronizace stavu systému.

7. Vyhodnocení ukazatelů spolehlivosti pro systém odolný proti poruchám implementovaný do FPGA využívající částečnou dynamickou rekonfiguraci pro opravu z poruchy a synchronizaci modulů TMR pro zotavení systému po poruše. Spolehlivostní parametry takto zabezpečeného systému by měly být porovnány s alternativním řešením využívajícím FG-TMR a synchronizující majoritní voliče. Dále by měly být vyhodnoceny navržené metody synchronizace z hlediska definovaných kritérií.

Následující kapitoly disertační práce popisují způsob splnění jednotlivých cílů disertační práce a dílčí výzkumné výsledky, kterých bylo v rámci vlastních výzkumných aktivit dosaženo.

Kapitola 5

Návrh opravovaného systému odolného proti poruchám

Tato kapitola popisuje návrh opravovaného systému odolného proti poruchám s využitím odolné architektury TMR a částečné dynamické rekonfigurace obvodu FPGA.

Kapitola volně navazuje na kapitolu 2, která čtenáři představila základní principy systémů odolných proti poruchám, a na kapitolu 3, ve které byl zhodnocen aktuální stav řešení problému obnovy stavu systému po poruše s využitím synchronizace stavu redundantních modulů architektury TMR. Cílem této kapitoly je být jakousi návrhářskou příručkou, diskutovat možnosti návrhu a implementace architektury TMR na různých úrovních abstrakce návrhu, představit návrhová schémata pro vytvoření rekonfigurovatelného systémů a popsat způsoby řízení dynamické rekonfigurace FPGA za běhu systému.

5.1 Zabezpečení systému pomocí TMR

Základní princip funkce a využití architektury TMR v systémech odolných proti poruchám byl popsán v kapitole 2.5.2. Implementace TMR v jednoduché formě, tedy při ztrojení zabezpečeného obvodu a přidání majoritního voliče na výstupy obvodů, není v praxi vždy dostačující. Důvodem je nechráněný majoritní volič, který představuje zranitelné místo v návrhu systému. Při návrhu systému zabezpečeného pomocí architektury TMR, která může být implementována s různou granularitou, je tedy nutné snažit se o minimalizaci takových zranitelných míst. V této kapitole jsou popsány praktické a pokročilejší metody pro integraci a implementaci TMR v návrhu obvodů.

5.1.1 Pokročilé metody implementace TMR

Způsoby implementace TMR mohou být rozděleny dle granularity s jakou je architektura TMR využita pro zabezpečení systému. Podle množství zabezpečené logiky v FPGA a zrnitosti vložení TMR lze rozlišovat mezi hrubozrnnou a jemnozrnnou strukturou architektury TMR. Hrubozrnné TMR označuje implementaci TMR na úrovni celého systému nebo funkčního modulu. Naopak jemnozrnné TMR označuje implementaci TMR na úrovni zabezpečení jednotlivých logických obvodů nebo registrů. V dalším textu práce bude použita pro rozlišení mezi TMR s různou granularitou následující konvence:

- Hrubozrnné TMR bude označeno jako *CG-TMR (Coarse Grained TMR)*.
- Jemnozrnné TMR bude označeno jako *FG-TMR (Fine Grained TMR)*.

Implementace hrubozrnné architektury TMR

Hlavním cílem implementace architektury TMR je zajištění bezporuchového běhu systému v případě poruchy jednoho z redundantních modulů v architektuře TMR. TMR je schopno maskovat jednu nebo více poruch typu SEU do té doby, dokud je ovlivněn pouze jeden z redundantních modulů.

S pomocí CG-TMR lze zabezpečit celý systém nebo velkou část obvodu najednou. Vytvoření architektury TMR, zahrnující ztrojení cílového systému a přivedení redundantních výstupů do majoritního voliče, je poměrně přímočarý postup. Nevýhoda hrubozrnné architektury se projeví při postupné akumulaci poruch typu SEU nebo výskytu několika poruch naráz. Pravděpodobnost poruchy redundantního modulu v architektuře CG-TMR roste s velikosti logiky FPGA, která je zabezpečena [53]. Majoritní volič umožňuje maskovat poruchu jednoho modulu architektury TMR a případně více poruch vzniklých v rámci jednoho modulu TMR. Problém nastává, jestliže se další porucha objeví v druhém modulu TMR. V takovém případě celý systém TMR selhává, protože již není možné zaručit prostřednictvím majoritního voliče správný výstup. Z tohoto důvodu se použití TMR kombinuje s technikami pro opravu poruch v konfigurační paměti FPGA (periodické čištění nebo modulární rekonfigurace).

Optimalizace hrubozrnné architektury TMR

Použití jemnějšího vkládání architektury TMR může být preferováno v případě předpokládané vyšší četnosti výskytu poruch typu SEU. Odolná architektura s jemnější architekturou TMR může být výhodnější díky vyšší schopnosti detekce, lokalizace a opravy vzniklých poruch [71] [95]. Výběrem větších nebo menších logických bloků, které jsou zabezpečeny pomocí TMR a implementací dílčích majoritních voličů lze najít kompromis v použití TMR a zmírnit nevýhody řešení využívajících maximální nebo minimální velikost modulu TMR. Akumulaci poruch v menších redundantních modulech lze více tolerovat, protože s použitím jemnějšího rozlišení zabezpečení obvodů pomocí architektury TMR klesá pravděpodobnost poruchy ve dvou redundantních instancích stejného modulu. Další výhodou tohoto přístupu je např. možnost klasifikovat typy poruch na základě místa nebo možných následků, které mohou být poruchou způsobeny [53].

Majoritní voliče mají významnou roli v technice implementace TMR, protože zamezují šíření následků poruchy z výstupní logiky obvodů. Vložením majoritních voličů na výstupy kombinačních nebo sekvenčních logických bloků lze vytvořit bariéry pro zachycení chyb způsobených poruchou. Takový přístup zahrnuje rozdělení návrhu systému do oddělených bloků a jejich zabezpečení pomocí TMR. Problémem je nalezení optimálního rozdělení obvodové logiky do majoritně volených logických bloků tak, aby se snížila pravděpodobnost poruchy v propojení dvou redundantních bloků, jež jsou voleny jedním majoritním voličem. Použití malých bloků vyžaduje implementaci velkého počtu majoritních voličů, což může být příliš nákladné z hlediska obvodové režie a ztráty výkonu. Na druhou stranu, implementace majoritních voličů pouze na výstupu zvyšuje pravděpodobnost poruchy dvou odlišných redundantních logických bloků [53]. Technice vkládání majoritních voličů do návrhu se blíže věnuje podkapitola 5.1.2.

Optimalizací implementace CG-TMR se zabývali např. autoři [80], kteří navrhli použití architektury TMR pouze pro zabezpečení obvodů, jež jsou více citlivé na poruchy typu SEU. Tento přístup je označován jako *STMR (Selective TMR)*. Přínosem STMR je, že za malou cenu ztráty odolnosti u méně citlivých obvodů na poruchy lze ušetřit velké procento zdrojů obvodů FPGA.

Implementace jemnozrnné architektury TMR

Implementace zabezpečení systému pomocí jemnozrnné architektury TMR se vyznačuje tím, že je snahou aplikovat techniku TMR na nejnižší logické úrovni návrhu. V tabulce 5.1 jsou porovnány významné parametry, výhody a nevýhody použití hrubozrnné a jemnozrnné architektury TMR.

Vlastnost	CG-TMR	FG-TMR
Maskování vícenásobných poruch	Ne *(lze maskovat pouze poruchy v jednom modulu)*	Ano *(lze maskovat poruchy ve více prvcích najednou)*
Schopnost lokalizace poruch	Malá	Velká
Vliv na max. hodinovou frekvenci	Malý	Velký *(vložení majoritních voličů do kritické cesty)*
Režie způsobená redundancí	Velká	Velká
Automatizovaná implementace	Spíše ano	Ano
Verifikace správné implementace	Jednoduchá	Složitá *(nutné využít speciální nástroje pro verifikaci)*

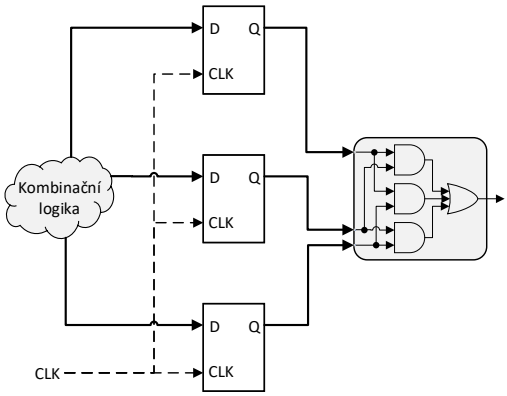
Tabulka 5.1: Porovnání hlavních rysů CG-TMR a FG-TMR

Podle rozsahu použití TMR pro zabezpečení logických zdrojů v kombinačních a sekvenčních obvodech, a způsobu vložení majoritních voličů do obvodové logiky, lze rozlišit několik typů implementace TMR.

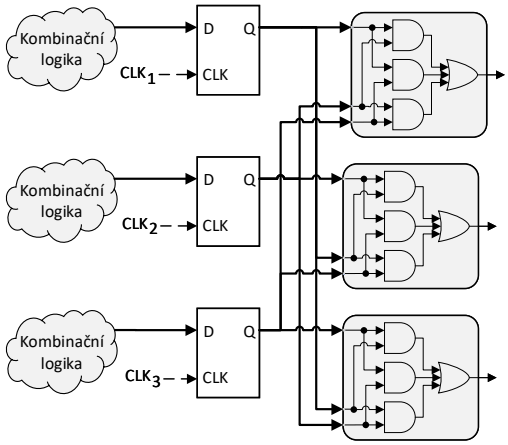
- *Lokální TMR (LTMR)* – Technika lokálního TMR spočívá ve ztrojení klopných obvodů typu D (obvody nazývané anglicky *flip-flop*), nacházejících se v logickém návrhu, a vložení majoritních voličů na jejich výstupy. Příklad zabezpečení logiky obvodů pomocí LTMR je znázorněn na obrázku 5.1. Přestože jsou klopné obvodů typu D chráněny proti vlivu poruch typu SEU za pomoci TMR, tyto obvody mohou stále zachytit poruchu typu SET vzniklou v kombinační logice vedoucí na vstup hradel. Porucha typu SET může mít ve výsledku stejný vliv jako porucha typu SEU a změnit stav klopného obvodu. LTMR může být použito pro obvody pracující na nízké hodinové frekvenci, u kterých lze předpokládat nižší pravděpodobnosti zachycení poruchy typu SET na vstupech klopných obvodů.

- *Globální TMR (GTMR)* – Technika globálního TMR spočívá ve ztrojení všech obvodových a logických zdrojů implementovaných v návrhu, včetně rozvodu hodinového signálu. GTMR může být aplikováno v návrhu ručně a nebo pomocí speciálních nástrojů jako je např. Xilinx X-TMR, Mentor Precision nebo BL-TMR. Příklad vložení GTMR v návrhu je znázorněn na obrázku 5.2. Každý klopný obvod typu D je zabezpečen pomocí TMR a majoritního voliče, který je taktéž ztrojen. Výstupy z majoritních voličů jsou přivedeny pomocí zpětné vazby zpět do registrů pro zajištění jejich správné hodnoty. Na výstupu jsou umístěny tzv. výstupní voliče implementované pomocí tří-stavového vyrovnávacího registru. Tento registr je nastaven na úroveň vysoké impedance v případě, že je výstupním voličem detekována nesprávná hodnota. V případě GTMR je jedinou zranitelnou části obvodu výstupní majoritní volič. Nicméně pokud jsou propojeny výstupy tří výstupních majoritních voličů, pak se takové pro-

pojení chová jako analogový majoritní volič, ve kterém dvě správné hodnoty logického signálu potlačují minoritní úroveň signálu.

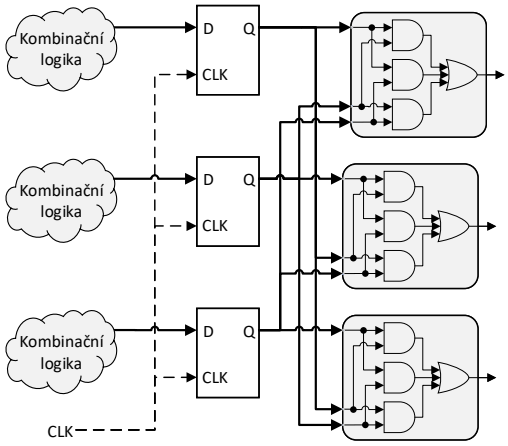


Obrázek 5.1: Lokální TMR



Obrázek 5.2: Globální TMR

- *Distribuované TMR (DTMR)* – Technika distribuovaného TMR, stejně jako GTMR, spočívá ve ztrojení všech obvodových a logických zdrojů implementovaných v návrhu. Oproti GTMR nejsou ztrojeny globální hodinové signály. Globální signály jsou sdíleny mezi redundantními instancemi klopných obvodů typu D. Příklad implementace DTMR je znázorněn na obrázku 5.3. Výhodou DTMR oproti GTMR je, že zde nehrozí chyby způsobené nerovnoměrnou distribucí hodinového signálu (tzv. *clock skew*).

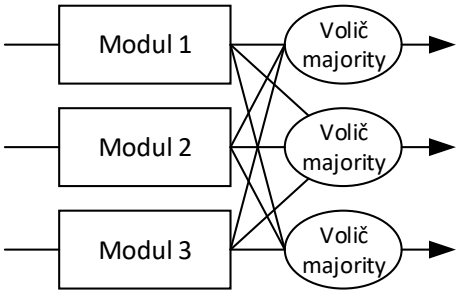


Obrázek 5.3: Distribuované TMR

5.1.2 Principy implementace a vkládání majoritního voliče

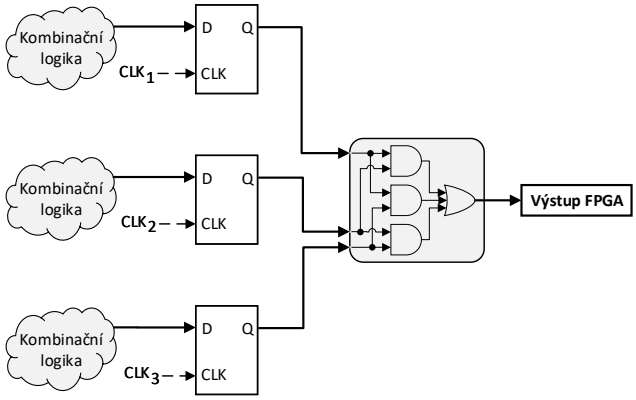
Při implementaci architektury TMR má klíčovou roli majoritní volič. Podle způsobu jeho vložení do logiky návrhu lze rozlišovat několik typů:

- *Ztrojený majoritní volič* – Nezabezpečený majoritní volič je kritickým a zranitelným místem v architektuře TMR. Z tohoto důvodu by měl být majoritní volič také ztrojen. Ztrojení majoritního voliče je vhodné u dvou vzájemně propojených logických bloků, kdy ztrojené výstupy jednoho bloku mohou snadno navázat na ztrojené vstupy dalšího bloku. Schéma zapojení ztrojeného majoritního voliče je znázorněno na obrázku 5.4.



Obrázek 5.4: Ztrojený majoritní volič

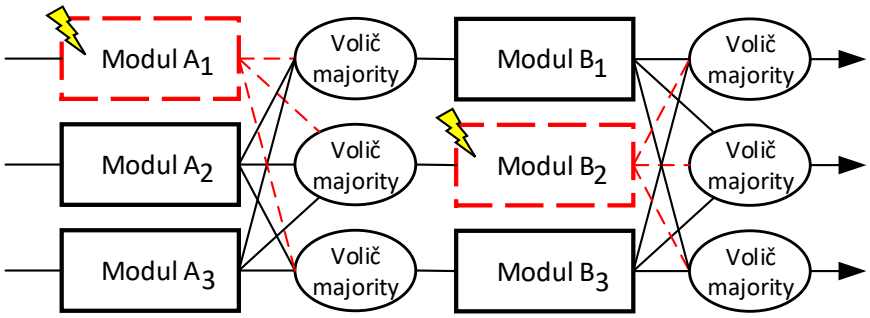
- *Redukující majoritní volič* – Tento typ majoritního voliče redukuje výstupy tří redundantních modulů TMR na jediný výstup. Redukující majoritní volič je základní typ majoritního voliče. Tento typ majoritního voliče se nejčastěji používá na výstupech obvodů. Alternativou může být použití externího obvodu pro výběr majority z výstupů produkovaných obvodem FPGA. Schéma redukujícího majoritního voliče je znázorněno na obrázku 5.5.



Obrázek 5.5: Redukující majoritní volič

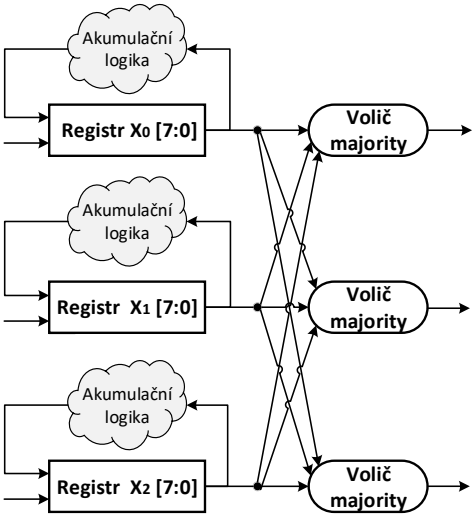
- *Oddělující majoritní volič* – Oddělující majoritní voliče jsou používány ke zvýšení spolehlivosti návrhu systému pomocí rozdělení návrhu do více oddělených bloků a jejich nezávislém zabezpečení pomocí TMR. Systém zabezpečený pomocí této techniky by měl být schopen bezporuchového provozu i v případě výskytu několika poruch. Poruchy v nezávislých blocích TMR jsou totiž maskovány na daleko jemnější úrovni návrhu systému. Díky izolaci poruch v rámci jednotlivých bloků TMR je možné také

zabránit nekontrolovanému šíření poruch uvnitř systému. Příklad využití oddělujících voličů je znázorněn na obrázku 5.6.

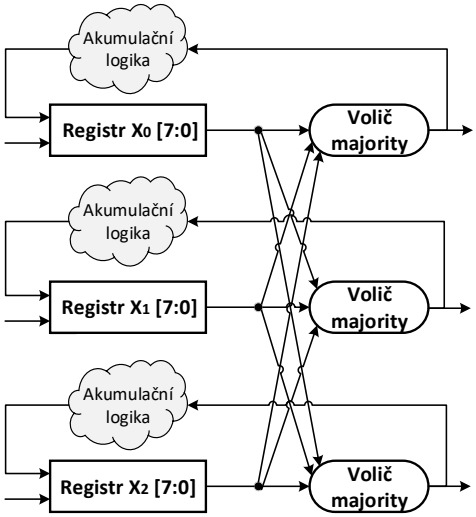


Obrázek 5.6: Oddělující majoritní volič

- *Synchronizující majoritní volič* – Cílem synchronizujících majoritních voličů je obnova správného stavu registrů (klopných obvodů typu D) v případě vzniku poruchy typu SEU. Porucha typu SEU může být z konfigurační paměti FPGA odstraněna pomocí rekonfigurace nebo periodického čištění konfigurační paměti. Nicméně nesprávná hodnota v registrech aplikační logiky zůstává do té doby, než je přepsána novou hodnotou nebo opravena. Problémem zůstává oprava stavu registrů, které mají přivedenu zpětnou vazbu. Zpětná vazba může způsobit, že nesprávná hodnota v registrech přetrvává. Tomuto efektu lze zabránit implementací tzv. synchronizujících majoritních voličů u registrů se zpětnými vazbami. Na obrázku 5.7 je znázorněna implementace majoritního voliče na výstupech z registrů, který není schopen zabránit akumulaci chybné hodnoty vlivem zpětné vazby registrů. Obrázek 5.8 znázorňuje implementaci synchronizujícího majoritního voliče v rámci zpětné vazby registrů.



Obrázek 5.7: Nesynchronizující volič



Obrázek 5.8: Synchronizující volič

- *Majoritní volič pro více časových domén* – Návrh systému může vyžadovat, že má každý z redundantních modulů architektury TMR přiveden dedikovaný hodinový signál. Dedikované hodinové signály mohou mít stejnou nebo různou frekvenci. V případě synchronního návrhu může docházet k časovému posunu. V případě využití asynchronních časových domén může každý z modulů TMR pracovat na jiné hodinové frekvenci. V takových případech jsou nutná speciální opatření a implementace mechanismu, který umožní spolehlivé porovnání výstupních signálu a jejích majoritní volení.

5.1.3 Automatické generování TMR

Implementace architektury TMR pro zabezpečení cílového systému se často provádí manuálně. Nicméně manuální implementace TMR může být často velmi zdlouhavá, náchylná na vznik implementačních chyb nebo v případě FG-TMR obtížně verifikovatelná. Nevýhody manuální implementace se projeví hlavně v případě implementace FG-TMR na úrovni RTL nebo na úrovni logických hradel. Z těchto důvodů vznikla řada nástrojů provádějících automatické generování TMR na různých úrovních návrhu, od FG-TMR po CG-TMR. Některé nástroje umožňují také implementaci jiných typů zabezpečení logiky FPGA (např. DMR).

Porovnání dostupných nástrojů pro automatické generování TMR

Porovnání vybraných dostupných nástrojů a metodik pro automatické vkládání TMR je uvedeno v tabulce 5.2. Firma Xilinx nabízí pro odolné obvody FPGA z rodiny Virtex-5QV a Virtex-4QV, které jsou určené pro letecké a vojenské systémy, komerční nástroj *TMRTool*[1], který umožňuje implementaci celkového zabezpečení navržené logiky pomocí GTMR (označované jako XTMR). Mezi další komerční nástroje se řadí *Mentor Precision Hi-Rel*[2] [63] a *Synopsys Synplify Premier*[3]. Tyto nástroje podporují velké množství obvodů FPGA od různých výrobců. Tyto nástroje implementují vlastní proces syntézy, což umožňuje optimalizovat vložení TMR s ohledem na další návrhová omezení.

Mezi nástroje, které jsou výsledkem výzkumných aktivit, se řadí *BL-TMR*[4], *TMRG (TMR Generator)*[5] [55], *TLegUp* [60]. Tyto nástroje neprovádí implementaci TMR během syntézy, nýbrž se soustředí na vložení TMR před syntézou do kódu HDL nebo na vložení TMR na úrovni netlistu. Nástroj TLegUP umožňuje automatické generování TMR pro obvody popsané pomocí jazyka C pro vysokoúrovňovou syntézu. Zabezpečený popis obvodu je transformován do popisu v HDL [60] [117] [2]. Nástroje TMRG provádí vložení TMR na úrovni popisu v jazyce Verilog [55].

Principy automatického generování TMR

Nástroj pro automatické generování TMR musí provést následující kroky: zanalyzovat návrh, vytvořit tři redundantní kopie pro každý zabezpečovaný obvod, replikovat propojovací vodiče ve smyslu originální propojovací logiky a vložit majoritní voliče pro výběr správného výstupu ze zabezpečeného obvodu. Způsoby vkládání TMR v rámci návrhového procesu

[1] <www.xilinx.com/products/design-tools/tmrtool.html>
[2] <www.mentor.com/products/fpga/synthesis/precision-hi-rel>
[3] <www.synopsys.com/implementation-and-signoff/fpga-based-design/high-reliability.html>
[4] <http://reliability.ee.byu.edu/edif>
[5] <http://tmrg.web.cern.ch/tmrg>

Nástroj	Společnost (Institut)	Licence	Úroveň návrhu	Typ TMR
TMRTool	Xilinx	Komerční	Syntéza	FG-TMR
Precision Hi-Rel	Mentor Graphics	Komerční	Syntéza	FG-TMR
Synplify Premier	Synopsys	Komerční	Syntéza	FG-TMR
BL-TMR	Univerzita Brigham Young	Akademická	Netlist (EDIF)	FG-TMR, CG-TMR
TMRG	CERN	GNU	HDL (Verilog)	CG-TMR
TLegUp	Univerzita UNSW Sydney	Akademická	HLS C	CG-TMR

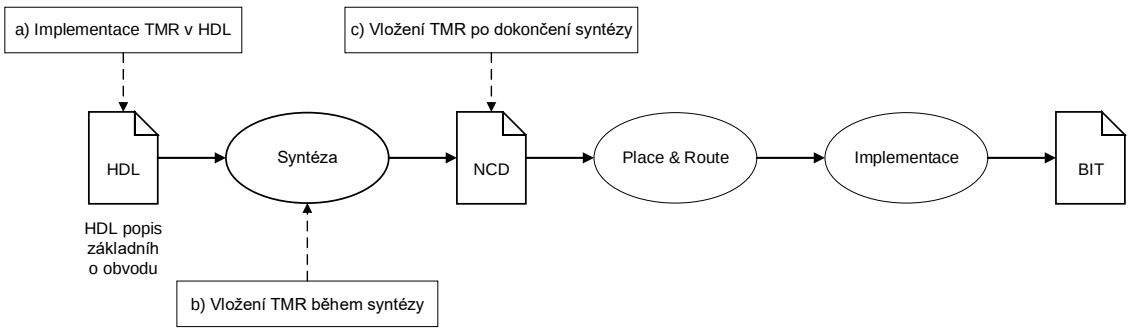
Tabulka 5.2: Přehled nástrojů pro automatické generování vkládání TMR

číslicového systému do FPGA jsou znázorněny na obrázku 5.9. Implementace architektury TMR může být provedena v několika bodech návrhového procesu:

a) Na úrovni vstupního HDL popisu (tedy např. úpravou popisu v jazyce VHDL nebo Verilog), před provedením syntézy.

b) Během procesu syntézy vstupního HDL popisu.

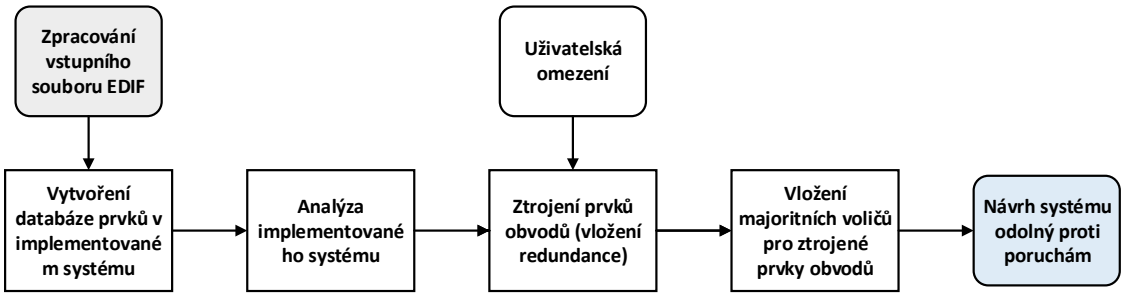
c) Po dokončení procesu syntézy na úrovni popisu jednotlivých hradel (netlistu) – Výhodou vložení TMR po dokončení netlistu je využití optimalizovaného netlistu. Jednou z výhod je fakt, že není nutné specifikovat dodatečná omezení pro redundantní logiku, jenž se snaží syntézní nástroje odstranit během optimalizace návrhu [12].



Obrázek 5.9: Způsoby vkládání TMR v rámci návrhového procesu

Nástroj BL-TMR

Nástroj BL-TMR (BYU-LANL Triple Modular Redundancy) umožňuje automatické vložení TMR na úrovni FG-TMR nebo CG-TMR pro část nebo celý zabezpečený systém. Nástroj BL-TMR provádí automatické generování TMR na úrovni popisu logických hradel, kdy vstupem je soubor netlist ve formátu EDIF vygenerovaný během syntézy popisu na úrovni HDL. Zjednodušený návrhový proces nástroje BL-TMR je znázorněn na obrázku 5.10.



Obrázek 5.10: Návrhový proces nástroje BL-TMR

Nejdříve jsou analyzovány vstupní data ve formátu EDIF. Z analyzovaných struktur logických obvodů je vytvořen graf, ve kterém jsou vyhledány silně souvislé komponenty. V rámci těchto podgrafů jsou identifikovány struktury obsahující zpětné vazby.

Na základě těchto informací je provedena klasifikace a rozdělení obvodových struktur do tří logických částí: obvodů se zpětnými vazbami, vstupními obvody (jež jsou připojeny do logických obvodů se zpětnými vazbami) a výstupní obvody. Následně je na základě požadavků návrháře provedeno automatické vložení TMR. Nástroj BL-TMR automaticky zjistí, zda je po ztrojení logických komponent obvodů nutné vložit majoritní volič nebo přidat zpětnou vazbu. Nově vytvořený popis zabezpečených logických obvodů je opět uložen do formátu EDIF. Výstup nástroje BL-TMR může být následně zpracován při fázi mapování na struktury cílové technologie [18] [47] [78].

Automatická implementace zabezpečení pomocí BL-TMR pro zvolený vstupní číslicový systém je široce konfigurovatelná. Nástroj BL-TMR umožňuje implementaci jak celkového tak částečného FG-TMR. Proces generování TMR může být také např. omezen limitem na 50% nebo 75% využití plochy zvoleného obvodu FPGA. Nástroj BL-TMR se skládá z několika kroků (resp. dílčích nástrojů), jež jsou vzájemně provázané. Každý z nástrojů provádí specifickou operaci nad aktuální reprezentací vstupního popisu obvodů [18].

1. *JEdifBuild* – JEdifBuild provádí prvotní zpracování vstupního popisu obvodů ze vstupních netlistů ve formátu EDF.

2. *JEdifAnalyze* – JEdifAnalyze provádí základní analýzu popisu obvodů se zaměřením na zpětné vazby a vstupně-výstupní bloky obvodů.

3. *JEdifNMRSelection* – JEdifNMRSelection určuje, která část vstupního popisu obvodů bude ztrojena. Tento nástroj může být spuštěn několikrát za účelem provedení částečného zabezpečení různých částí číslicového systému. Nástroj podporuje kromě zabezpečení pomocí TMR také implementaci duplexního systému.

4. *JEdifVoterSelection* – JEdifVoterSelection určuje místa ve vstupním popisu obvodů, do kterých bude vložen majoritní volič. Pro implementaci majoritních voličů lze vybrat některý z několika optimalizovaných algoritmů, které umožňují např. vložit majoritní volič před každý nebo za každý klopný obvod typu D.

5. *JEdifMoreFrequentVoting* – JEdifMoreFrequentVoting je volitelný krok v procesu generování TMR, který umožňuje implementaci dodatečných oddělujících majoritních voličů. Oddělující majoritní voliče umožňují rozdělení číslicového návrhu na několik oddělených zabezpečených celků.

6. *JEdifNMR* – JEdifNMR provádí ztrojení logických zdrojů obvodů a vložení majoritních voličů na základě strategie určené uživatelem.

Nástroj BL-TMR je v této disertační práci využit pro vygenerování alternativního způsobu zabezpečení s využitím FG-TMR. Zabezpečený systém je podroben experimentům, při kterých jsou do plochy konfigurační paměti FPGA vloženy poruchy typu SEU. Spolehlivostní parametry pro zabezpečení systému pomocí FG-TMR a CG-TMR s využitím rekonfigurace a synchronizace stavu jsou porovnány a vyhodnoceny v kapitole 8.

5.2 Principy návrhu rekonfigurovatelného systému v FPGA

Cílem této kapitoly je popsat principy a důležité detaily návrhu rekonfigurovatelného systému v FPGA. Možnosti rekonfigurace obvodů FPGA s konfigurační pamětí typu SRAM od firmy Xilinx byly již popsány v kapitole 2.3.4. Tato kapitola se soustředí na popis dílčích kroků procesu návrhu funkčního rekonfigurovatelného systému.

5.2.1 Rekonfigurovatelný návrh systému

Návrh rekonfigurovatelného systému v FPGA, zjednodušeně řečeno, spočívá v rozdělení plochy FPGA na jednu statickou oblast a na jednu nebo více rekonfigurovatelných oblastí a v implementaci řadiče, který bude zodpovědný za provedení rekonfigurace. Při návrhu rekonfigurovatelného systému lze postupovat takto (body 1. – 4. a body 5. – 6. jsou podrobněji popsány v podkapitolách 5.2.2 a 5.3):

1. Návrh logiky rekonfigurovatelného systému.
2. Oddělená syntéza pro statickou část a rekonfigurovatelné moduly.
3. Plánování rozmístění – při plánování rozmístění komponent číslicových obvodů v rekonfigurovatelném návrhu lze využít následující dva přístupy:
   a) Využití standardního postupu pro rekonfigurovatelný návrh – Při plánování rozmístění je provedeno rozdělení logiky číslicového systému na statickou a dynamickou část. V konfigurační paměti jsou vytvořeny rekonfigurovatelné regiony *PRR (Partial Reconfigurable Region)*, do kterých může být logika rekonfigurovatelných modulů naprogramována. Během následné implementace je pro každý PRM a příslušný PRR vytvořen částečný bitstream. Výhodou tohoto přístupu je, že vývojové nástroje mohou optimalizovat využití zdrojů obvodu FPGA a časová omezení napříč rozhraními všech rekonfigurovatelných modulů [50].
   b) Využití relokace bitstreamu – Pokud nejsou zavedeny žádná omezení na vygenerování částečných konfiguračních souborů pro rekonfigurovatelné moduly PRM, nástroje od firmy Xilinx vytvoří pro každý PRM a každou variantu logického obvodu, který má být umístěn v daném PRM, jiný částečný konfigurační bitstream. To znamená, že pro M rekonfigurovatelných modulů PRM a N variant implementované logiky v PRM je nutné vygenerovat $M \times N$ částečných konfiguračních bitstreamů [44]. Tento způsob vygenerování konfiguračních bitstreamů s sebou nese režii na uchování všech souborů v externí paměti. Nicméně způsob generování částečným bitstreamů lze optimalizovat tak, aby bylo možné použít jednu konfiguraci pro danou variantu implementované logiky v různých

rekonfigurovatelných modulech. Pro N variant implementované logiky by tedy bylo vygenerováno opět jen N konfiguračních bitstreamů. Technika, která vede k takto použitelným konfiguračním bitstreamům, se nazývá *relokace bitstreamů*.

4. Implementace rekonfigurovatelného systému – Dokončení fáze rozmístění a propojení logických prvků číslicového návrhu a provedení fáze vygenerování úplného a částečných konfiguračních bitstreamů.

5. Návrh způsobu uchování částečných bitstreamů pro provedení rekonfigurace (např. v externí paměti Flash).

6. Návrh řadiče rekonfigurace.

5.2.2 Návrhový proces pro obvody FPGA od firmy Xilinx

Návrhový proces pro implementaci číslicového systému do obvodu FPGA od firmy Xilinx byl zobecněně popsán v kapitole 2.3.3. Cílem této kapitoly je navázat na základní návrhový proces a popsat aspekty procesu z hlediska návrhu a implementace rekonfigurovatelného systému. Firma Xilinx pro obvody FPGA Virtex-5[6] nabízí grafické vývojové prostředí *ISE Design Suite* a *PlanAhead*. Alternativou je využití jednotlivých nástrojů zodpovědných za provedení dílčích kroků v návrhovém procesu a jejich spuštění z příkazového řádku nebo pomocí naprogramovaného skriptu s využitím jazyka Tcl či jiného. Tento přístup je preferovaný v případě, že návrhář chce mít jednotlivé kroky v návrhovém procesu přizpůsobené vlastním potřebám a jednoduše opakovatelné.

Fáze logické syntézy

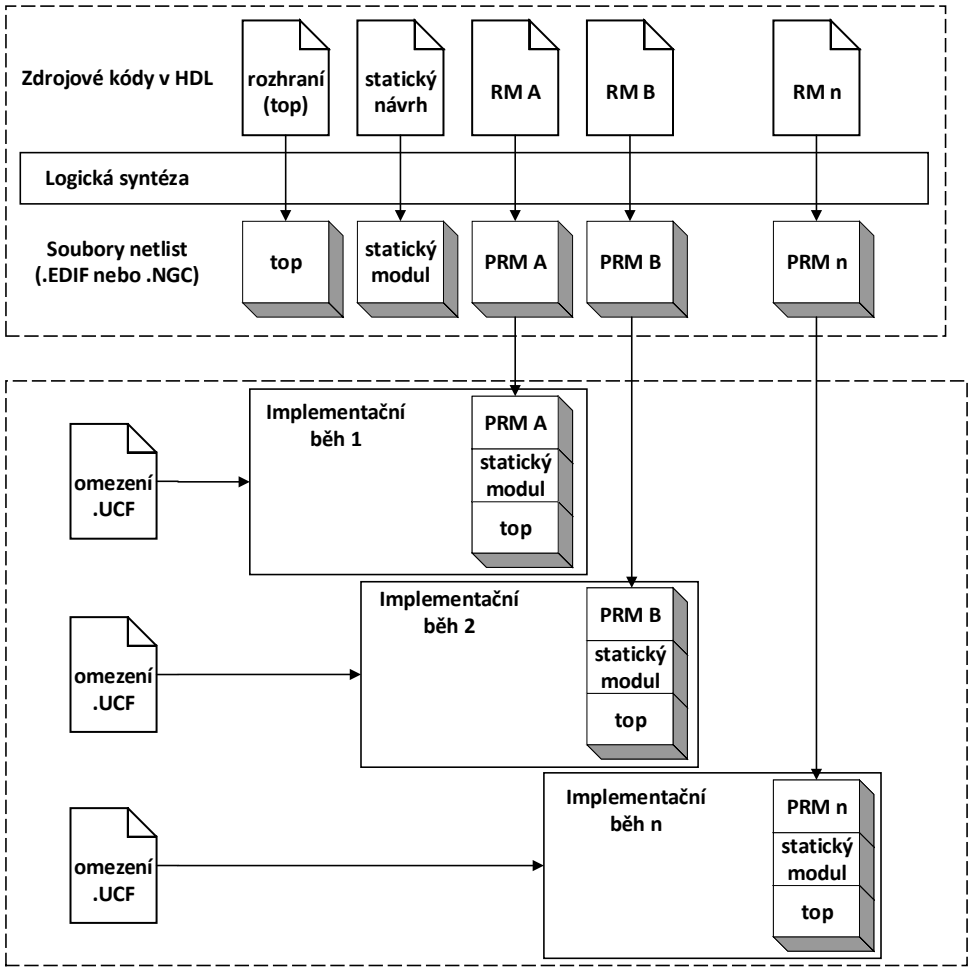
Rekonfigurovatelný návrh vyžaduje provést syntézu návrhu zdola nahoru. Při syntéze návrhu zdola nahoru je cílový návrh složen z nezávislých komponent. V případě návrhu rekonfigurovatelného systému je nutné vytvořit pro každý rekonfigurovatelný oddíl oddělený netlist. Vytvoření nezávislých netlistů pro každý rekonfigurovatelný modul může být dosaženo např. pomocí nezávislých projektů. Při syntéze modulu na nejvyšší úrovni jsou jednotlivé oddíly zpracovány jako tzv. black-box [106]. Statická logika může být syntetizována společně do jednoho nebo několika netlistů.

Syntéza popisu obvodů v jazyce HDL je provedena za pomocí nástroje XST [112]. Důležité parametry a omezení použité pro syntézu komponent rekonfigurovatelného návrhu jsou následující:

- Parametr `-keep_hierarchy` povoluje syntézu hierarchického návrhu. Tento způsob syntézy je nutný při implementaci návrhu složeného z několika oddělených komponent (netlistů).

- Parametr `-iobuf` umožňuje povolit nebo zakázat vkládání vyrovnávacích pamětí na logické vstupy nebo výstupy. Pro syntézu návrhu na nejvyšší úrovni může být vkládání vyrovnávacích pamětí povoleno. Nicméně pro úspěšnou syntézu odděleného modulu, ať už se jedná o rekonfigurovatelný modul nebo pouze samostatnou komponentu, musí být vkládání zakázáno.

[6] Podobně lze postupovat i v případě návrhu rekonfigurovatelného systému pro novější obvody FPGA s použitím vývojového prostředí Vivado.



Obrázek 5.11: Návrhový proces rekonfigurovatelného systému pro obvody od firmy Xilinx

V případě implementace architektury TMR je nutné využít následující parametr pro omezení syntézy, jenž může vlivem optimalizace způsobit odstranění redundantních obvodů.

- Parametr `-equivalent_register_removal` umožňuje povolit nebo zakázat optimalizaci využití klopných obvodů typu D a jejich odstranění. Pro správnou syntézu systému využívající architekturu TMR pro zabezpečení musí být tento parametr deaktivován.

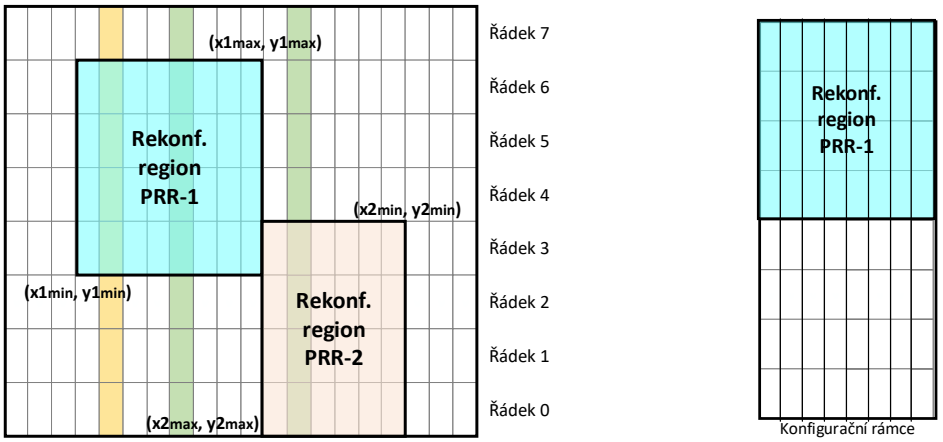
Aplikace omezujících podmínek návrhu

Omezující podmínky pro specifické kroky implementace číslicového systému mohou být zapsány pomocí speciálních příkazů v souboru *UCF (User Constraint File)* [104]. Pomocí UCF souboru je provedeno mapování mezi fyzickými piny na rozhraní obvodu FPGA a signály implementovaného číslicového návrhu. Omezující podmínky pro procesy mapování, rozmístění a propojení budou blíže popsány v následujících podkapitolách.

Fáze plánování rozmístění (floorplanning)

Pro nerekonfigurovatelný statický návrh lze ponechat plánování rozmístění návrhu uvnitř FPGA v režii nástroje PAR, který provede rozmístění a propojení logiky automaticky s ohledem na optimální rozložení a minimalizaci časových zpoždění. Vlastní plánování rozmístění prvků ve statickém návrhu může být provedeno ve speciálních případech, např. při nutnosti vysoké optimalizace využití prostředků FPGA nebo pro splnění přísných časových omezení.

Pro rekonfigurovatelný návrh číslicového systému v FPGA musí návrhář manuálně navrhnout rekonfigurovatelné oblasti, přidělit logické prvky návrhu definovaným oblastem a určit ty logické moduly návrhu, které mají být rekonfigurovatelné. Proces plánování rozmístění pro rekonfigurovatelný návrh začíná rozdělením implementované logiky číslicového návrhu do skupin a určením fyzických oblastí v rámci konfigurační paměti FPGA, kde budou tyto skupiny logických obvodů umístěny. Tyto oblasti se označují termínem *pblock (physical block)*. Do každé fyzické oblastí může být přiřazena libovolná část logiky číslicového návrhu. Seskupením logických obvodů, jež spolu komunikují nebo jsou součástí jednoho většího modulu, může být omezena délka propojovacích vodičů a sníženo zpoždění signálů [108]. Pro definování fyzické oblasti pblock je nutné použít omezující podmínky typu AREA_GROUP. Každý rekonfigurovatelný modul PRM musí být definován jako samostatný pblock. Nově definované fyzické oblasti v konfigurační paměti FPGA by měly mít obdélníkový charakter. Velikost fyzické oblasti je určena souřadnicemi (x0,y0,x1,y1), podobně jak je znázorněno na obrázku 5.12a.



(a) Omezení oblasti PRR

(b) PRR vs. konfigurační rámce

Obrázek 5.12: Vytvoření rekonfigurovatelných regionů v konfigurační paměti FPGA

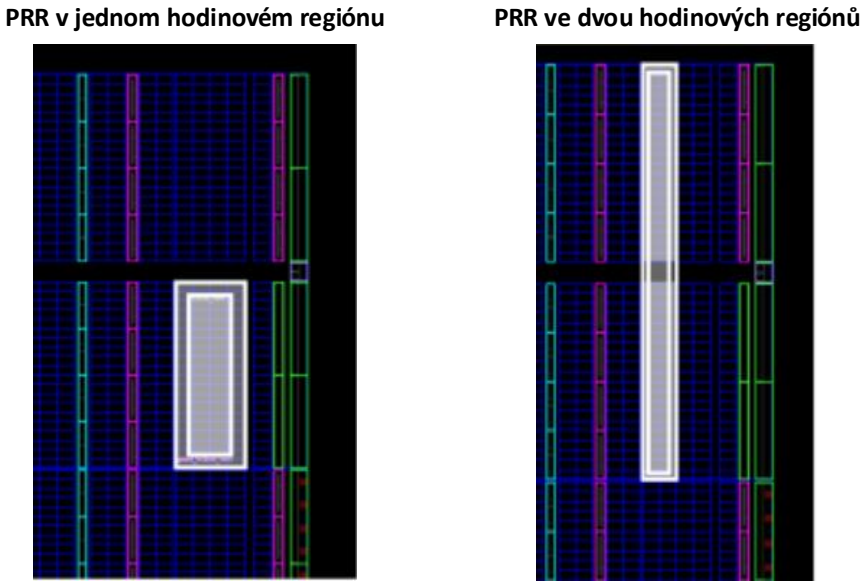
Příklad zápisu definice rekonfigurovatelné oblasti `pblock_reconfigurable_module` pro instanci logického modulu `reconfigurable_module` a přiřazení zdrojů obvodu FPGA (bloky SLICE, DSP a RAM) do této oblasti je uveden ve výpisu kódu 5.1.

```
INST "reconfigurable_module" AREA_GROUP = "pblock_reconfigurable_module";
AREA_GROUP "pblock_reconfigurable_module" RANGE=SLICE_X8Y40:SLICE_X27Y59;
AREA_GROUP "pblock_reconfigurable_module" RANGE=DSP48_X0Y16:DSP48_X2Y23;
AREA_GROUP "pblock_reconfigurable_module" RANGE=RAMB36_X0Y8:RAMB36_X1Y11;
```

Výpis 5.1: Ukázka definice rekonfigurovatelné oblasti v souboru UCF

Při návrhu rekonfigurovatelného systému musí být brán v potaz fakt, že některé vestavěné bloky a obvodové zdroje uvnitř obvodu FPGA nemohou být rekonfigurovány. Typickým příkladem je interní konfigurační port ICAP, který se vždy musí nacházet v rámci statické části návrhu. Logické komponenty, které musí zůstat statické jsou: bloky ovlivňující hodinový signál (MMCM, DCM, PLL, PMCD), globální hodinové vyrovnávací paměti (BUFG), I/O komponenty (IOLOGIC, IODELAY, IDELAYCTRL), speciální funkční bloky (např. BSCAN, ICAP, STARTUP, PCIE). Naopak, rekonfigurovatelné moduly mohou obsahovat pouze následující komponenty: LUT, flip-flop, paměti (bloková RAM, distribuovaná RAM, posuvné registry v LUT), bloky DSP.

Dále musí návrhář dbát na správné zarovnání rekonfigurovatelných oblastí uvnitř FPGA. Při definici rekonfigurovatelných fyzických oblastí je nutné zohlednit strukturu konfigurační paměti a vzít v potaz, že konfigurační rámec je nejmenší rekonfigurovatelná jednotka. Konfigurační rámce nemohou být sdíleny mezi rekonfigurovatelnými oblastmi. Doporučený postup je provést zarovnání rekonfigurovatelných oblastní na hranice konfiguračních rámců. V případě znázorněném na obrázku 5.12b , kdy část rekonfigurovaného konfiguračního rámce obsahuje logiku statického návrhu, je konfigurační rámec přepsán stejnými daty. Konfigurační paměť FPGA je rozdělena na hodinové regiony. Obrázek 5.13 znázorňuje umístění rekonfigurovatelné oblasti do jednoho a více hodinových regionů. Preferovaný způsob je umístit rekonfigurovatelnou oblast do jednoho regionu.



Obrázek 5.13: Příklad vytvoření PRR v jednom a nebo více hodinových regionech FPGA

Fáze verifikace rekonfigurovatelného návrhu

Pro ověření, že byl rekonfigurovatelný návrh správně sestaven a že mezi statickou a dynamickou části neexistují žádné konflikty, by měl být podle [106] spuštěn nástroj `pr_verify`. Tento nástroj provádí ověření konfigurací rekonfigurovatelného návrhu. Verifikaci konfigurací lze provést také přímo z nástroje PlanAhead.

Fáze generování bitstreamu pro konfiguraci FPGA

Konfigurační bitstream může být úplný nebo částečný. Úplný konfigurační bitstream konfiguruje celou konfigurační paměť FPGA a je použit pro statický návrh nebo pro rekonfigurovatelný návrh při počáteční konfiguraci FPGA. Částečný bitstream odpovídá pouze části konfigurační paměti, jež byla zvolena jako rekonfigurovatelný region v nástroji PlanAhead během fáze plánování rozmístění, a implementuje logiku PRM [41].

Při generování bitstreamu pro rekonfigurovatelný návrh by měly být pro program `bitgen` nastaveny následující parametry:

- Nastavení parametru `ActiveReconfig` na hodnotu `Yes` umožňuje předejít resetu FPGA při částečné dynamické rekonfiguraci. (Díky tomuto parametru není vložen příkaz GSR do výsledného částečného konfiguračního bitstreamu.)
- Nastavení parametru `Binary` na hodnotu `Yes` způsobí, že výsledný částečný konfigurační bitstream obsahuje pouze konfigurační data (soubor s příponou .BIN). Bez tohoto parametru by výsledný bitstream (soubor s příponou .BIT) obsahoval také hlavičku s dodatečnými informacemi.

Další parametry jsou popsány v [106]. Při použití návrhového postupu pro modulární částečnou rekonfiguraci by neměl být použit příkaz `-r`, který je podporován pouze v případě rozdílové částečné rekonfigurace. Rozdíly mezi těmito dvěma technikami využití částečné dynamické rekonfigurace byly již popsány v kapitole 2.3.4.

5.3 Oprava stavu systému pomocí rekonfigurace FPGA

Opravu poruchy v konfigurační paměti FPGA způsobené vlivem SEU lze provést pomocí rekonfigurace FPGA. Základní principy využití rekonfigurace pro opravu stavu systému byly popsány v kapitole 2.6. Pro implementaci systému odolného proti poruchám do obvodu FPGA s konfigurační paměti typu SRAM lze předpokládat využití obvodové redundance (např. některého typu TMR) a jedné ze tří hlavních technik pro opravu poruch.

a) *Periodické čištění konfigurační paměti FPGA* – Způsoby provádění periodického čištění a pokročilejší techniky pro periodickou kontrolu a opravu poruch v konfigurační paměti byly popsány v kapitole 2.6.1.

b) *Oprava trvalých poruch pomocí relokace částečných bitstreamů* – Tato technika byla představena v kapitole 2.6.2. Pro umožnění relokace částečných bitstreamů je nutné využít speciální metodiku, která zabezpečí možnost záměny rekonfigurovatelných modulů nezávisle na zvoleném rekonfigurovatelném regionu v konfigurační paměti FPGA.

c) *Rekonfigurace redundantní části systému* – Tento způsob opravy poruch předpokládá využití obvodové redundance pro detekci a maskování vlivu poruch a použití rekonfigurace pro přeprogramování PRM, ve kterém byla detekována porucha. Přeprogramování PRM může proběhnout při zastavení funkce systému nebo v případě architektury TMR za běhu systému.

Tato disertační práce se zaměřuje na poslední uvedený způsob opravy poruch v konfigurační paměti FPGA. Lze předpokládat, že pro systémy, pro které je důležitá schopnost co nejrychlejšího obnovení stavu systému do stavu bezporuchového, bude využita technika

rekonfigurace spolu s implementací architektury CG-TMR. Díky této redundantní architektuře je odolný systém schopen překonat situaci, kdy je v jednom z redundantních modulů detekována porucha a vrátit se do bezporuchového stavu, pokud je daný redundantní modul rekonfigurován v okamžiku detekce poruchy. V tabulce 5.3 jsou porovnány dvě techniky a) a c) s ohledem na využití CG-TMR nebo FG-TMR. (Tyto techniky mohou být také kombinovány s technikou pro opravu trvalých poruch pomocí relokace částečných bitstreamů.)

Vlastnost	Rekonfigurace modulů CG-TMR	Periodické čištění + CG-TMR | FG-TMR
Oprava za běhu systému	Ano	Ano
Latence opravy poruchy	Rychlá	Periodická oprava
Detekce latentních poruch	Ne	Ano
Synchronizace stavu	Explicitní	Explicitní | Implicitní
Navýšení spotřeby energie	Pouze při opravě	Periodický přírůstek

Tabulka 5.3: Porovnání metod pro opravu stavu systému pomocí rekonfigurace FPGA

Nevýhodou periodického čištění může být právě periodické provádění, protože oprava poruchy může být provedena až v další periodě čištění konfigurační paměti. Naopak při použití rekonfigurace modulů TMR může být porucha opravena ihned, jakmile je detekována. Nicméně tato technika neumožňuje detekci latentních (skrytých) poruch, jež mohou být naopak detekovány pomocí periodického čištění. Aby mohla být technika využívající rekonfiguraci modulů CG-TMR srovnatelná s periodickým čištěním a zabezpečením pomocí FG-TMR (obsahující synchronizující majoritní voliče), musí být implementován mechanismus synchronizace stavu mezi rekonfigurovatelnými moduly.

Současný stav poznáni v této oblasti, zahrnující techniky synchronizace na různých úrovních implementace TMR, byl již popsán v kapitole 3. Vlastní metodika pro návrh metod synchronizace stavu je představena v kapitolách 7 a 8.

Ať už je v praxi zvolena jakákoliv z výše zmíněných metod, je vždy nutné implementovat řídicí systém, v tomto případě tzv. řadič rekonfigurace, který bude řídit rekonfiguraci a provádět algoritmus opravy poruchy. V případě periodického čištění se takový systém nazývá scrubber. Nicméně základní požadavky na řízení rekonfigurace jsou podobné.

Řadič rekonfigurace je obvod zodpovědný za řízení celého procesu PDR, od čtení konfiguračního bitstreamu, zpracování dat konfiguračních rámců a přípravu dat pro rekonfiguraci, adresování cílového rámce v konfigurační paměti FPGA, programovaní konfigurační paměti přes některé z programovacích rozhraní (např. ICAP), po provádění doplňujících funkcí týkajících se detekce a klasifikace poruch, periodického provádění rekonfigurace nebo opravy poruchy ve zvoleném redundantním modulu architektury TMR.

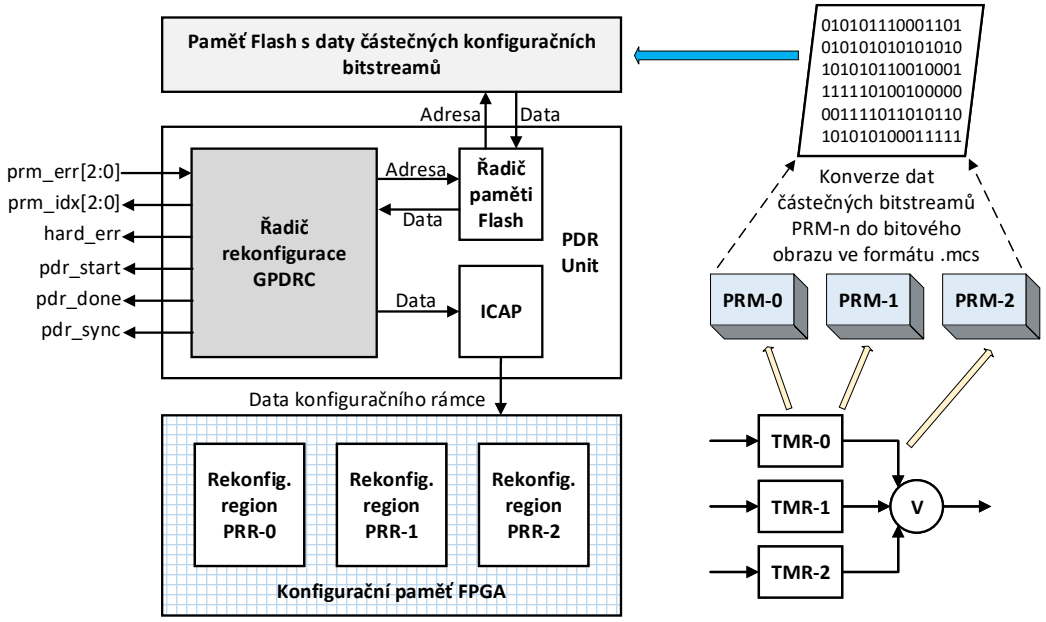
Tato disertační práce využívá řadič rekonfigurace *GPDRC (Generic Partial Dynamic Reconfiguration Controller)*, který byl vyvinut v rámci aktivit [87] [65] naši výzkumné skupiny, zabývající se spolehlivými systémy, na UPSY, FIT VUT v Brně.

5.3.1 Řadič rekonfigurace GPDRC

Řadič rekonfigurace je funkční jednotka, která využívá pro realizaci částečné dynamické rekonfigurace FPGA vnitřní konfigurační rozhraní ICAP. Tato jednotka může být syntetizována a v podobě netlistu integrována do jakéhokoliv návrhu číslicového systému v FPGA od firmy Xilinx. Řadič GPDRC je schopen pracovat na frekvenci 100 MHz. Předpokladem

je, že je řadič GPDRC umístěn ve statické části návrhu číslicového systému v FPGA. (Případně může být GPDRC taktéž zabezpečen a v případě poruchy rekonfigurován. Řadič může být implementován jako samoopravný systém nebo rekonfigurován pomocí externího řadiče.) Pro rekonfiguraci je potřebné vygenerovat částečné konfigurační bitstreamy pro jednotlivé PRM a uložit je do externí paměti (např. do paměti Flash). Pro řízení rozhraní ICAP využívá řadič GPDRC komponentu ICAP_VIRTEX5 dodanou v rámci nástrojů od firmy Xilinx. Funkce a implementace komponenty řadiče GPDRC byly popsány blíže v publikacích jeho autorů [87] [65].

Řadič rekonfigurace GPDRC byl v této disertační práci integrován spolu s řadičem paměti Flash a komponentou ICAP_VIRTEX5 do jedné funkční jednotky nazvané PDR Unit. Schéma jednotky PDR Unit a její použití je znázorněno na obrázku 5.14.



Obrázek 5.14: Jednotka pro řízení částečné dynamické rekonfigurace

Spuštění rekonfigurace je provedeno pomocí vstupní sběrnice *prm_err*. Po sběrnici prm_err je přenášen kód typu 1 z N, kde log. 1 označuje, který PRM má být rekonfigurován. (V případě CG-TMR, který modul TMR má být rekonfigurován.) Řadič rekonfigurace GPDRC disponuje rozhraním pro komunikaci s pamětí Flash a konfiguračním rozhraním ICAP, se kterými byl vzájemně propojen. Při experimentech, popsaných dále v této disertační práci, byla využita vývojová deska ML506 s obvodem FPGA Virtex-5 od firmy Xilinx. Tato vývojová deska obsahuje také paralelní Flash paměť[7], která je využita pro uložení binárních dat částečných konfiguračních bitstreamů. Řadič GPDRC musí být syntetizován s předinicializovanou vyhledávací tabulkou, která obsahuje adresy jednotlivých částečných bitstreamů. V případě aktivace rekonfigurace pro vybraný PRM pomocí sběrnice prm_err je nejdříve z vyhledávací tabulky vyčtena adresa začátku bitstreamu. Následně provádí vnitřní řídicí automat řadiče GPDRC vyčítání dat bitstreamu z paměti Flash a jejich zpracování. Při zpracování bitstreamu je nejdříve vyčtena hlavička bitstreamu, následně všechna data

[7]Paralelní paměť Flash je označována jako tzv. BPI Flash. Jedná se o paměť typu NOR. Paměť je k FPGA připojena pomocí BPI (Byte Peripheral Interface) rozhraní.

konfiguračních rámců a patička ukončující bitstream. Řídicí automat GPDRC taktéž řídí komunikaci s vnitřním konfiguračním rozhraním ICAP přes komponentu ICAP_VIRTEX5. Konfigurační data vyčtená z paměti Flash jsou přes vyrovnávací paměť FIFO zapsána na vstup jednotky ICAP. Dále řadič GPDRC umožňuje indikaci aktuálně rekonfigurovaného PRM pomocí výstupu *PRM_index* a požadavku na začátek synchronizace pomocí výstupu *sync*. Pokud je řadič GPDRC připojen přímo k jednotce detekující poruchy v odolné architektuře (např. majoritní volič architektury TMR), pak je řadič schopen provádět automaticky rekonfiguraci PRM, ve kterém byla detekována porucha. Řadič GPDRC je dále schopen detekovat taktéž permanentní poruchy, které mohou být rozpoznány na základě opakované detekce poruchy ve stejném PRM. Detekovaná trvalá porucha je indikována pomocí výstupu *hard_err*.

Režie spotřebovaných zdrojů FPGA při syntéze jednotky PDR Unit pro Virtex-5 na vývojové desce ML-506 s ISE Design Suite (verze 14.7) je uvedena v tabulce 5.4.

Virtex5 XC5VSX50T Implementované komponenty	Počet registrů	Počet LUT	Počet BMEM	Počet DMEM
Řadič rekonfigurace GPDRC	281	327	1	0
Řadič paměti Flash	3	5	0	0
Ostatní logika	0	39	0	0
Jednotka PDR Unit celkem	284	371	1	0

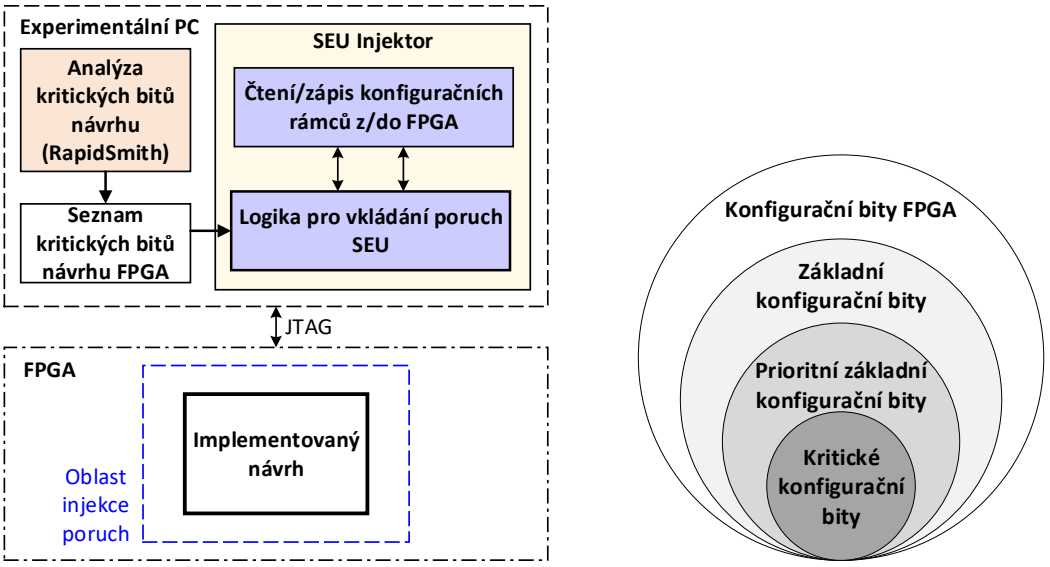
Tabulka 5.4: Režie implementace jednotky PDR Unit a řadiče rekonfigurace GPDRC

5.4 Simulace poruch v konfigurační paměti FPGA pomocí injektoru poruch typu SEU

Při návrhu systému opravitelného z poruchy je nutné dbát také na dostatečné ověření správné funkce opravného mechanismu. V případě opravitelného systému implementovaného v SRAM FPGA, který využívá rekonfiguraci pro odstranění poruch z konfigurační paměti FPGA, je simulace poruch typu SEU v konfigurační paměti FPGA jednou z možností pro ověření správné funkce systému.

Při experimentech provedených v rámci této disertační práce byl využit SEU framework, který byl vyvinut v rámci aktivit naši výzkumné skupiny [88]. Schéma injektoru poruch typu SEU je znázorněno na obrázku 5.15a. Tento nástroj umožňuje injektovat poruchy typu SEU (překlopení hodnoty bitu) do konfigurační paměti za běhu systému pomocí částečné dynamické rekonfigurace prováděné přes rozhraní JTAG. Poruchy mohou být umístěny cíleně, případně je lze generovat náhodně po celé ploše konfigurační paměti, čímž lze také simulovat projev poruchy typu MBU. Umístění generovaných poruch lze upřesnit za pomocí detailnější analýzy rozložení cílového systému v konfigurační paměti FPGA a využití logických prostředků. Pro analýzu lze využít nástroj RapidSmith[8] [59]. Cílem analýzy konfiguračních bitů číslicového návrhu je nalezení tzv. kritických bitů. Tyto bity ovlivňují přímo funkci implementované logiky uvnitř konfiguračních bloků FPGA. Na obrázku 5.15b je znázorněna množina kritických bitů jako podmnožina všech konfiguračních bitů obvodu FPGA a základních bitů implementovaného číslicového návrhu. Díky analýze kritických bitů lze optimalizovat proces simulace poruch a urychlit prováděné experimenty.

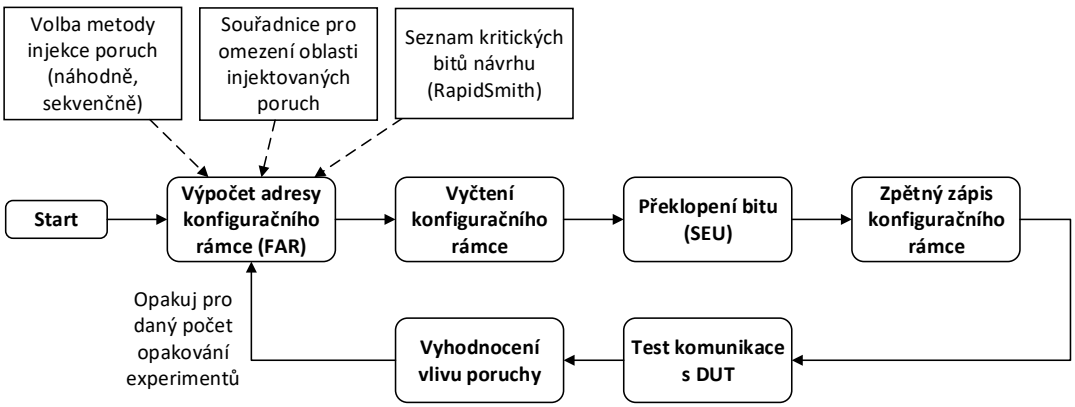
[8] http://rapidsmith.sourceforge.net



(a) Schéma injektoru poruch typu SEU (b) Typy konfiguračních bitů v FPGA

Obrázek 5.15: Principy simulace poruch v konfigurační paměti FPGA

Koncept experimentů využívající injekci poruch je znázorněn na schématu 5.16. Tímto způsobem byly provedeny experimenty popsané v kapitolách 7 a 8. Při provedených experimentech byly injektovány poruchy do oblasti konfigurační paměti FPGA, ve které se nacházel zabezpečený systém. Během injekce poruch byla neustále udržována komunikace pomocí specifické komunikační sběrnice (tedy sběrnice CAN nebo UART) mezi experimentálním PC a testovaným návrhem zabezpečeného systému (na obrázku 5.16 označen jako DUT). Cyklus injekce poruch je ukončen v momentě ztráty komunikace s DUT. V tomto momentě lze předpokládat selhání funkce systému, které je způsobeno injektovanou poruchou nebo akumulací všech předešlých poruch injektovaných od začátku testovacího cyklu.



Obrázek 5.16: Provedení experimentů s injekcí poruch SEU

Kapitola 6

Metodika synchronizace stavu rekonfigurovatelných modulů

Tato kapitola tvoří jádro disertační práce. Je zde popsána vytvořená metodika pro návrh metod synchronizace stavu rekonfigurovatelných modulů TMR a související problematika návrhu kompletního systému odolného proti poruchám s využitím této nové metodiky.

6.1 Návrh systému s využitím metodiky synchronizace stavu

V této kapitole je popsán způsob využití nové metodiky a její zařazení do celého návrhového procesu systému odolného proti poruchám.

6.1.1 Aplikace metodiky během návrhového procesu systému

Samotná aplikace metodiky synchronizace stavu rekonfigurovatelných modulů musí být při návrhu systému odolného proti poruchám kombinována s dalšími technikami, metodikou pro návrh architektury TMR a metodikou pro návrh rekonfigurovatelného systému v FPGA. Metodika synchronizace stavu může být aplikována na již existující návrh rekonfigurovatelné architektury TMR. Nicméně návrh systému musí být pro umožnění synchronizace stavu mezi rekonfigurovatelnými moduly opět upraven. Z tohoto hlediska je lepší provést souběžný návrh systému odolného proti poruchám s využitím všech tří metodik. V této podkapitole bude dále popsán způsob souběžného návrhu.

Vstupy metodiky

Vstupy metodiky jsou následující:

- Popis nezabezpečeného systému v jazyce HDL.
- Uživatelská omezení pro implementaci systému do FPGA (tzn. omezení pro vlastní implementaci, omezení z hlediska vykonávaní funkce systému v reálném čase).
- Předpokládané hodnoty ukazatelů spolehlivosti pro zabezpečený systém (předpokládaná hodnota MTBF, požadovaná dostupnost systému).
- Uživatelská omezení a parametry pro návrh metody synchronizace.

Postup návrhu systému s využitím metodiky

Návrhový proces systému předpokládá splnění následujících kroků:

1. Návrh architektury TMR – Tento krok zahrnuje volbu typu architektury TMR pro zabezpečení cílového systému, rozdělení návrhu nezabezpečeného systému a určení nezávisle zabezpečených oddílů návrhu systému, implementace logiky pro majoritní volení a detekci poruch v TMR. Při návrhu architektury TMR může být implementována také dodatečná logika, která bude součástí každého z redundantních modulů. Metodika zabezpečení systému pomocí TMR byla blíže popsána v kapitole 5.1.

2. Ověření správné funkce navržené architektury TMR bez logických obvodů realizujících opravu nebo synchronizaci stavu systému po poruše.

3. Návrh rekonfigurovatelného systému – Při návrhu rekonfigurovatelného systému je nutné definovat rekonfigurovatelné oblasti v konfigurační paměti, přiřadit implementovanou logiku každého z modulů architektury TMR do rekonfigurovatelných modulů PRM a implementovat řadič rekonfigurace pro řízení procesu PDR a opravy poruchy v konfigurační paměti náležící modulům PRM. Návrhový proces pro vytvoření rekonfigurovatelného systému byl blíže popsán v kapitole 5.2.

4. Ověření správné funkce rekonfigurovatelného systému a procesu rekonfigurace. Protože je stav rekonfigurovaného modulu nekonzistentní se zbytkem systému, je pro úspěšné ověření provedené rekonfigurace nutné provést opětovnou resynchronizaci stavu všech modulů. Nejjednodušším způsobem synchronizace rekonfigurovatelných modulů je prostřednictvím globálního nulování.

5. Návrh specifické metody synchronizace s využitím metodiky pro synchronizaci stavu a implementace logiky pro řízení a provedení synchronizace. Samotný návrh metody synchronizace je upřesněn v následujících kapitolách.

6. Ověření[1] správné funkce rekonfigurace a synchronizace stavu systému během výskytu poruch v konfigurační paměti FPGA. Poruchy v systému mohou být simulovány manuálně, vložením chybových testovacích vektorů na vstupy logických obvodů, a nebo automaticky, pomocí injekce poruch do konfigurační paměti FPGA.

Výstupy metodiky

Výstupy metodiky jsou následující:

- Návrh systému odolného proti poruchám zabezpečeného pomocí rekonfigurovatelné architektury TMR, ve které je každý z rekonfigurovatelných modulů PRM obnovitelný prostřednictvím navržené metody synchronizace stavu.
- Konfigurační bitstream pro počáteční konfiguraci obvodu FPGA.
- Částečné bitstreamy pro rekonfigurovatelné moduly PRM.

[1] Ověření správné funkce systému odolného proti poruchám probíhá v praxi často ve specializovaných laboratořích, kde jsou provedeny radiační testy, během nichž je implementovaný systém vystaven reálnému vlivu nabitých částic.

6.1.2 Výběr metody synchronizace

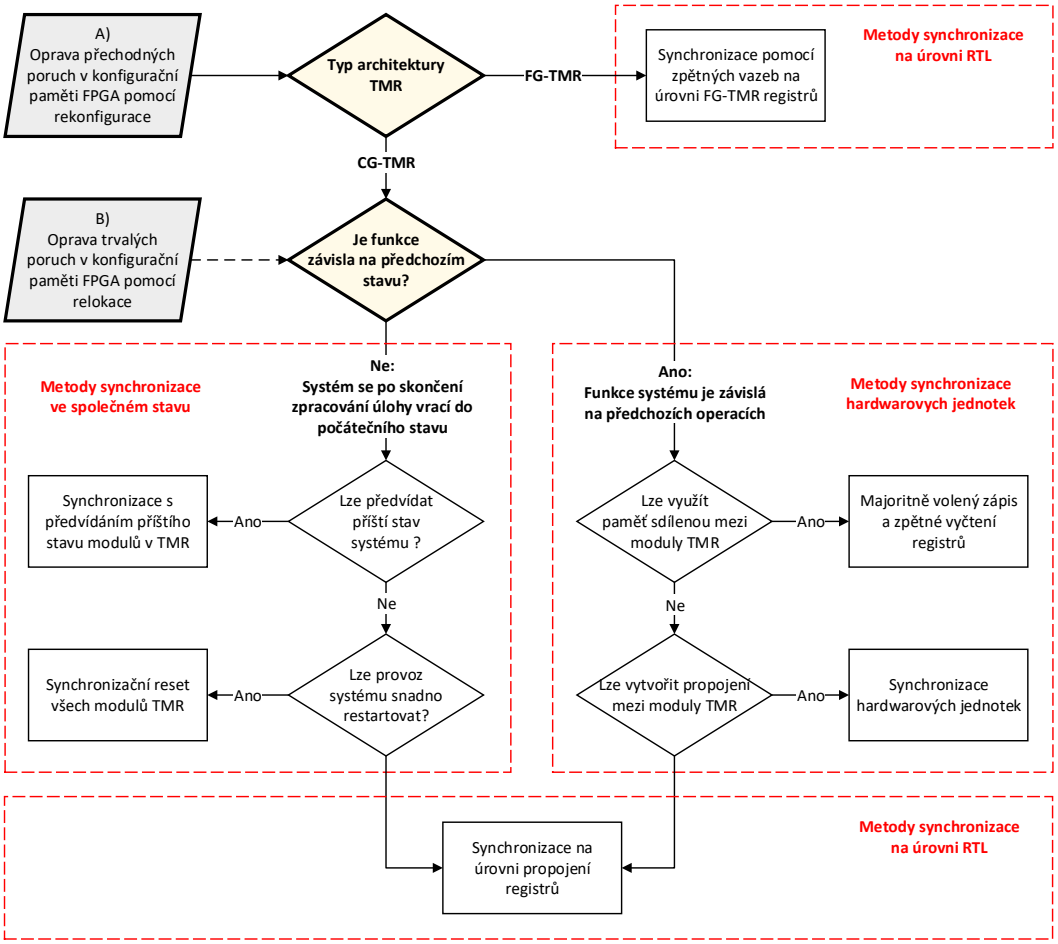
Na základě vyhodnocení vhodnosti různých způsobů synchronizace pro různé typy systémů v kapitole 3.6, lze rozdělit metody synchronizace do tří hlavních kategorií:

1. *Metody synchronizace ve společném stavu* – Mezi metody synchronizace ve společném stavu lze zařadit synchronizaci pomocí nulování jednotek a synchronizaci pomocí předvídání stavu v systémech řízených konečným automatem.

2. *Metody synchronizace na úrovni RTL* – Metody provádějící synchronizaci na úrovni RTL využívají fyzické propojení mezi registry, které může být implementováno ručně nebo automaticky. Do této skupiny může být zařazena také technika popsána v kapitole 5.1.2, jenž implementuje synchronizující majoritní voliče na úrovni FG-TMR.

3. *Metody synchronizace na úrovni redundantních modulů* – Tyto metody jsou zaměřeny na synchronizaci stavu mezi redundantními moduly systému, při které jsou využity systémové prostředky (např. komunikační sběrnice nebo sdílená paměť).

Způsob zvolení vhodné metody synchronizace zohledňující výše uvedené rozdělení metod je znázorněn na obrázku 6.1.



Obrázek 6.1: Výběr metody synchronizace stavu

Při výběru metody synchronizace stavu systému zabezpečeného pomocí TMR je nutné v první řadě zohlednit typ použité architektury TMR. Metodika navržená v této disertační práci se soustředí na způsoby synchronizace v rekonfigurovatelné architektuře CG-TMR.

Nicméně při klasifikaci metod synchronizace je nutné vzít v úvahu i jiné způsoby zabezpečení systému. Příkladem je použití synchronizujících voličů při implementaci FG-TMR. Nástroj BL-TMR [18] pro automatické generování FG-TMR byl popsán v kapitole 5.1.3.

Dále je nutné zmínit návaznost metody synchronizace stavu na celkový proces opravy a obnovy stavu systému odolného proti poruchám. Požadavek na synchronizaci stavu systému zabezpečeného pomocí TMR může nastat v několika případech:

a) Stav redundantních modulů TMR je synchronizován po dokončení opravy přechodné poruchy. Tato disertační práce dále uvažuje pouze tento způsob opravy poruch.

b) Stav redundantních modulů TMR je synchronizován v případě provedení relokace částečného bitstreamu do jiného rekonfigurovatelného regionu z důvodu opravy trvalé poruchy v konfigurační paměti FPGA.

c) Stav redundantních modulů TMR je synchronizován pro opravu chyb akumulovaných v systému bez použití rekonfigurace. Po odeznění vlivu přechodné poruchy mohou být v sekvenčních obvodech modulu, v němž byla rozpoznána porucha, stále uloženy chybné hodnoty. Synchronizace může být použita pro přepis těchto hodnot.

Největší vliv na výběr a návrh optimální metody synchronizace stavu má charakter synchronizačních objektů, které mají být synchronizovány. Pro systémy, které se vždy vrátí do výchozího stavu a které nemají funkci závislou na předchozích výpočtech, může být dostačující synchronizovat pouze provádění funkce systému. U těchto systémů nejsou synchronizovány data stavu, nýbrž samotný stav systému (většinou v rámci stavového automatu), na základě kterého je provedena jeho inicializace a jsou nastaveny výstupy. Naopak komplexnější systémy, jejichž funkce závisí na historii předešlých výpočtů a které obsahují množství registrů sekvenční logiky, mohou vyžadovat implementaci sdíleného přístupu do paměti nebo využití synchronizační sběrnice pro vzájemnou komunikaci mezi moduly. Jestliže se pro daný systém nehodí ani jedna z předešlých možností, je nutné implementovat specifické hardwarové řešení na úrovni propojení všech registrů, které je nutné synchronizovat.

6.2 Principy návrhu metod synchronizace stavu

Pro návrh vhodné metody synchronizace stavu rekonfigurovatelných modulů v systému TMR existuje několik dílčích problémů, které je nutné vyřešit na základě analýzy cílového systému a systémových požadavků. Tyto problémy jsou popsány v následujícím textu:

1. Výběr vhodného stavu, ve kterém bude synchronizace hardwarových jednotek v systému TMR provedena. Tento stav bude z pohledu metodiky dále označován jako *synchronizační stav*.

2. Určení dat stavu systému, která je nutné synchronizovat. Paměťové elementy (zejména registry sekvenční logiky) obsahující data stavu systému budou z pohledu metodiky dále nazývány jako *synchronizované objekty*.

3. Nalezení vhodné synchronizační sekvence a implementace vzájemného propojení mezi redundantními hardwarovými jednotkami, které umožní provedení synchronizace stavu opravené instance modulu s ostatními bezporuchovými moduly.

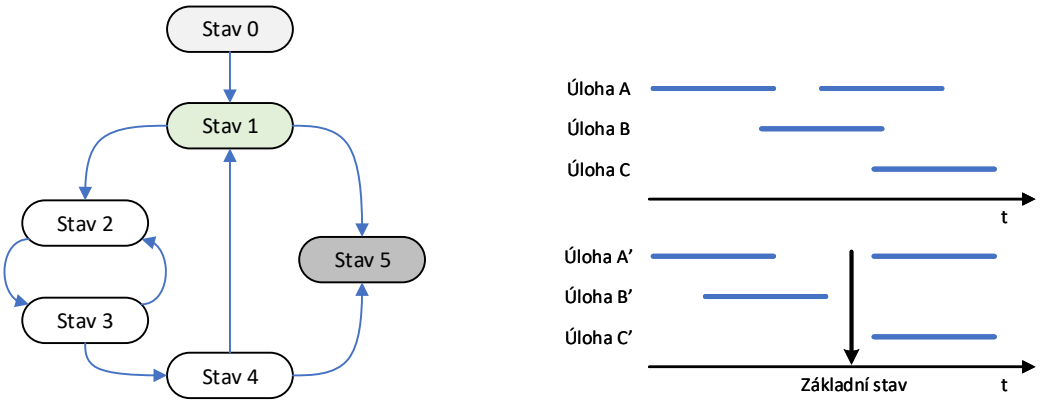
6.2.1 Krok 1: Výběr vhodného synchronizačního stavu

Při stanovení stavu systému určeného k synchronizaci kontextu stavu musí být zohledněna proveditelnost implementace samotné synchronizace, požadavky pro práci v reálném čase, zachování integrity a konzistence dat stavu. Zvolený synchronizační stav by měl splňovat následující požadavky:

- *Dostupnost* – Vlastnost stavu systému udávající, zda je systém schopný spolehlivě a deterministicky dosáhnout daného stavu v určitém čase. Tato vlastnost se vztahuje spíše k systémům založeným na konečných automatech. Pro komplexní systémy je důležitější skutečnost, zda lze v daném stavu pozastavit činnost systému.
- *Konzistence* – Vlastnost stavu systému udávající, zda je stav systému v daný okamžik konzistentní a zda lze provést kopii celého datového kontextu systému bez narušení integrity dat. Synchronizaci lze provést se zachováním integrity všech dat (řada vzájemně spjatých parametrů a registrů v systému může být pravidelně aktualizována, tudíž synchronizace těchto dat musí být provedena atomicky).
- *Minimálnost* – Vlastnost stavu systému udávající, že je množina synchronizovaných objektů v daném stavu minimální. Výběrem minimálního synchronizačního stavu lze zaručit optimální návrh synchronizace, při kterém je využití zdrojů obvodu FPGA a doba provedení synchronizace omezena pouze na synchronizaci nejnutnějších objektů.

Na obrázku 6.2a je znázorněn příklad stavového automatu systému, pro který by měl být navržen způsob synchronizace. Počáteční stav 0 a koncový stav 5 jsou stavy, které nesplňují požadavek na dostupnost. Ve stavech 2 a 3 se mohou provádět výpočty nebo zpracování dat. Tyto stavy nejsou vhodné k synchronizaci, protože obsahují nekonzistentní data. Ve stavu 4 může být provedeno odeslání dat přes výstupní rozhraní. Stav 4 může obsahovat data, která pozdější už nejsou potřebná. Proto se jako nejvhodnější stav pro synchronizaci jeví stav 1, který splňuje většinu požadavků pro synchronizační stav.

Obrázek 6.2b znázorňuje provádění úloh procesorem v reálném čase. Na tomto obrázku je znázorněn princip nalezení tzv. *základního stavu (ground state)* [52], který umožňuje provedení synchronizace v okamžiku, kdy není prováděna žádná z programových úloh.



(a) Jednoduchý stavový automat

(b) Provádění úloh v procesoru

Obrázek 6.2: Výběr vhodného synchronizačního stavu

6.2.2 Krok 2: Definice synchronizovaných objektů stavu systému

Stav systému je reprezentován hodnotami všech paměťových elementů hardwarových jednotek uvnitř systému. Paměťové elementy v sekvenční logice mohou být implementovány pomocí dvou základních typů obvodů:

a) *Asynchronní klopný obvod RS* (tzv. latch) – Klopný obvod typu RS má dva vstupní signály R a S, které slouží k nulování výstupu klopného obvodu (vstup R – Reset) a nastavení hodnoty zachycené klopným obvodem (vstup S – Set).

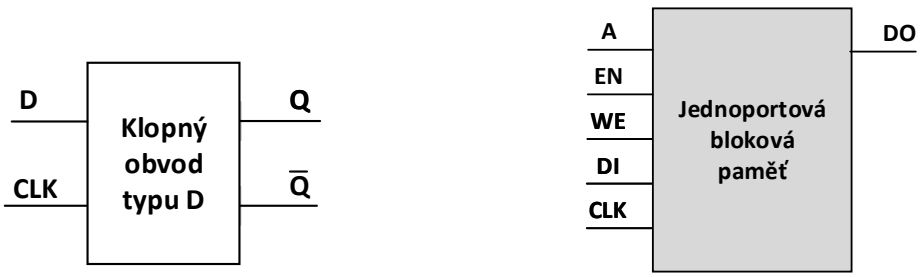
b) *Synchronní klopný obvod D* – Klopný obvod typu D přenáší vstupní informaci ze vstupu D na výstup Q s příchodem náběžné hrany hodinového signálu.

Na základě klopného obvodu typu D mohou být implementovány další typy sekvenčních obvodů. Seskupením několika klopných obvodů typu D lze vytvořit n-bitové registry, posuvné registry nebo čítače. Tyto sekvenční obvody mohou obsahovat další dodatečné vstupy, umožňující např. reakci na další vstupní signály nebo trvalé nastavení hodnoty obvodu. Blokové schéma klopného obvodu typu D je znázorněn na obrázku 7.3c.

Klopné obvody typu D, které si pamatují nastavenou hodnotu po dobu jednoho hodinového cyklu nemusí být synchronizovány. Jejich hodnota, která by byla v případě poruchy typu SEU změněna na nesprávnou, je opět přenastavena s příchodem dalšího hodinového cyklu. Hlavní problém nastává v případě, že má paměťový prvek přivedenu zpětnou vazbu.

Kromě paměťových prvků založených na bázi klopných obvodů jsou v obvodech FPGA často využívány blokové paměti BRAM. Tyto paměti jsou implementovány pomocí primitiv integrovaných v FPGA. Základní schéma jednoportové paměti BRAM je znázorněno na obrázku 7.3d. Opět platí, že pokud tyto paměti obsahují data potřebná pro provoz systému, je nutná jejich synchronizace.

Pro návrh metody synchronizace stavu je nutné znát umístění a funkci všech paměťových elementů v systému (dále jen synchronizačních objektů), které jsou důležité z pohledu přenesení funkční kopie stavu jednoho redundantního modulu do druhého, tak aby byl druhý modul schopen následně pokračovat ve stejné funkci jako první modul. Paměťové elementy, které obsahují pouze mezivýsledky a jsou při každém výpočtu přepsány jinou hodnotou, není nutné synchronizovat. Naopak, paměťové elementy důležité pro funkci systému nebo obsahující aplikační data je nezbytné synchronizovat.



(a) Klopný obvod typu D (b) Paměť BRAM

Obrázek 6.3: Základní typy paměťových prvků v FPGA

6.2.3 Krok 3: Návrh propojení hardwarových jednotek

Kromě nalezení vhodného synchronizačního stavu a definice všech synchronizovaných objektů v návrhu systému, je nutné navrhnout vzájemné propojení redundantních modulů tak, aby nebyla implementace příliš náročná (s ohledem na spotřebované zdroje FPGA) a zároveň, aby byla synchronizace provedena v co nejkratším čase. Propojení redundantních modulů pro synchronizaci jejich stavů lze realizovat několika způsoby:

a) *Propojení pro synchronizaci ve společném bodě* – Jestliže stav všech synchronizovaných objektů závisí pouze na stavu redundantních modulů systému, ve kterém se právě nachází, pak lze implementovat propojení, které umožní současný přechod synchronizovaného modulu a ostatních modulů do stejného stavu.

   - *Společné nulování modulů* – Provedení synchronizovaného restartu všech tří redundantních modulů je nejjednodušší způsob synchronizace. Pro správnou funkci by měly mít všechny sekvenční obvody nulovací vstup, který zaručí jejich opětovnou reinicializaci.
   - *Přenesení provozního stavu* – Pro propojení, které umožní přenesení provozního stavu z jednoho modulu do druhého, je nutné implementovat logiku pro vyčtení stavu a pro nastavení nového stavu. Logika pro vyčtení stavu je potřebná pro vyčkání na zvolený kontrolní stav, do kterého modul systému přejde. Aktivace logiky pro nastavení nového stavu se provede ve chvíli, kdy logika pro vyčtení stavu detekuje očekávaný stav. Stav synchronizovaného modulu je následně v rámci jednoho hodinového cyklu přenastaven na stav zdrojového modulu.

b) *Propojení na úrovni registrů*

   - *Paralelní propojení* – Propojení paralelními vodiči umožňuje velmi rychlou synchronizaci, synchronizovaná data jsou přenášena paralelně. Nevýhodou je nutnost použití velkého počtu vodičů. V případě jednoduchého obvodu lze propojit všechny registry za pomocí paralelních vodičů a provést synchronizaci v rámci jednoho hodinového cyklu. V případě složitějšího systému je nutné implementovat multiplexované propojení. To přináší určité zlepšení v náročnosti propojení systémů pomocí paralelních vodičů. Je nutné implementovat logiku, která bude adresovat specifické registry pomocí multiplexoru.
   - *Sériové propojení* – Sériové propojení má nejméně náročnou implementaci, která spočívá ve zřetězeném propojení všech synchronizovaných registrů a vytvoření pomyslného posuvného registru, který má jeden sériový datový vstup a jeden sériový datový výstup.

c) *Propojení na úrovni modulů*

   - *Propojení redundantních systémů přes centralizované úložiště dat* – Příkladem může být sdílená paměť BRAM se synchronizovaným přístupem ke čtení a zápisu dat pomocí výběru z majority.
   - *Propojení pomocí sběrnice* – Propojení pomocí sběrnice má nejmenší nároky na propojovací vodiče ale největší na použitou logiku, která umožní adresovat datové registry systému. Dá se chápat jako pokročilejší implementace paralelního multiplexovaného propojení.

6.3 Základní architektura synchronizačních obvodů

Číslicové systémy implementované v FPGA se obecně skládají z mnoha různých komponent. Často je návrh systému tvořen kombinací sekvenční logiky, registrů a pamětí, kombinační logiky, konečných automatů, případně měkkého jádra procesoru nebo jiných funkčních komponent. Z tohoto důvodu musí být proces synchronizace řízen na dvou různých úrovních, na úrovni redundantních modulu a na úrovni dílčích komponent každého z modulů. Na základě této úvahy, publikované v [94], bylo pro realizaci synchronizace stavu v návrhu systému navrženo několik základních typů obvodů:

Arbitr synchronizace – Arbitr je obvodová jednotka, která řídí celý průběh synchronizace na nejvyšší úrovni systému. Arbitr synchronizace je aktivován řadičem rekonfigurace ihned po dokončení rekonfigurace oblasti PRM, ve které byla detekována porucha. Základní požadavky na arbitr synchronizace jsou následující:

1. Arbitr určuje, který redundantní modul má být synchronizován a jaká bude funkce ostatních redundantních modulů v TMR během synchronizace.
2. Arbitr komunikuje se všemi řadiči synchronizace a řídí proces synchronizace.
3. Arbitr přepíná funkci redundantních modulů do normálního provozního stavu po dokončení synchronizace.

Synchronizační sběrnice – Logické propojení mezi redundantními moduly, které obsahuje řídicí a datové signály a umožňuje komunikaci mezi všemi obvodovými jednotkami během synchronizace.

Řadič synchronizace – Řadič je obvodová jednotka zodpovědná za řízení synchronizace vnitřních komponent uvnitř každého redundantního modulu systému. Základní úlohy řadiče synchronizace jsou následující:

1. Řadič komunikuje s arbitrem synchronizace během procesu synchronizace.
2. Řadič adresuje jednotlivé komponenty a registry uvnitř redundantního modulu.
3. Řadič provádí, vzhledem k roli přiřazené arbitrem danému modulu v TMR, několik činností během procesu synchronizace:
   a) řídí přenos stavových informací v referenčním obvodě,
   b) řídí příjem stavových informací v synchronizovaném obvodě a ukládání dat do vnitřních registrů obvodu,
   c) pozastavení funkce jednotek, které nemají žádnou roli v procesu synchronizace do jeho dokončení.
4. Řadič sleduje stav referenční obvodové jednotky. V případě, že se systém dostane do stavu určeného pro synchronizaci, řadič informuje arbitra synchronizace, který může následně rozhodnout o dalším kroku synchronizace.

Synchronizační rozhraní – Pro synchronizaci na úrovni registrů je nutné implementovat rozhraní použité k propojení jednotlivých komponent v systému TMR pomocí synchronizační sběrnice.

Popsaná architekura nemusí být při implementaci synchronizace přesně dodržena. V některých případech může být funkce arbitra a řadiče synchronizace integrována do jedné funkční jednotky. Specifická funkce těchto dílčích komponent bude vždy záviset na zvolené metodě synchronizace a vlastnostech cílového systému.

6.4 Parametry metod synchronizace stavu

Návrh a implementace vybrané metody synchronizace jsou úzce spjaty s architekturou cílového systému a jeho funkcí. Návrh logiky synchronizace dále musí zohlednit různé požadavky na spolehlivost, režii implementace, příkon, rychlost a další aplikačně specifická omezení. Samotná implementace logiky pro synchronizaci sekvenčních obvodů má výrazný vliv na funkci systému a jeho parametry. Tyto parametry lze rozdělit dle jejich charakteru na dynamické a statické.

6.4.1 Dynamické parametry

Dynamické parametry reflektují vliv synchronizace na provoz a funkci systému:

- *Vliv na funkci systému* – Synchronizace by měla být provedena ve vhodném synchronizačním stavu, který splňuje požadavek na konzistenci stavu. Dosažení takového stavu může vyžadovat krátké pozastavení funkce systému. To následně umožní bezpečné přenesení synchronizovaných objektů mezi instancemi redundantních modulů. Lze tedy říci, že synchronizace je blokující. U blokujících metod je proto snahou dosáhnout co nejkratší doby synchronizace a minimálního ovlivnění funkce systému.

  Aby bylo možné hodnotit vliv synchronizace na chování systému, bylo definováno následující rozdělení metod pro synchronizaci stavu:

  - *Neblokující metody* – Metody, u kterých není nutné pozastavit ani omezit provoz systému pro jejich synchronizaci. Příkladem neblokující metody může být technika synchronizace v konečných automatech s předvídáním budoucího stavu [9] popsaná v kapitole 3.4.
  - *Online metody* – Metody, které umožňují provedení synchronizace za běhu systému ve speciálním režimu nebo programové úloze.
  - *Blokující metody* – Metody, u kterých provedení synchronizace vyžaduje zastavení běhu systému. Příkladem blokujících metod mohou být metody pro synchronizaci stavu v měkkém jádru procesoru [96] [43] [42] [32] [6] [66] [67] [34] [35].

- *Čas potřebný pro provedení synchronizace* – Doba synchronizace je závislá na zvolené implementaci metody synchronizace a na objemu dat, která mají být synchronizována. Protože je časová náročnost provedení synchronizace přímo úměrná objemu synchronizačních objektů, synchronizační stav by měl splňovat požadavek na minimálnost.

6.4.2 Statické parametry

Statické parametry mají sice nepřímý vliv na funkci systému, ale jsou úzce spjaty s implementovaným mechanismem pro synchronizaci stavu systému. Za nejdůležitější statické parametry lze pokládat následující parametry:

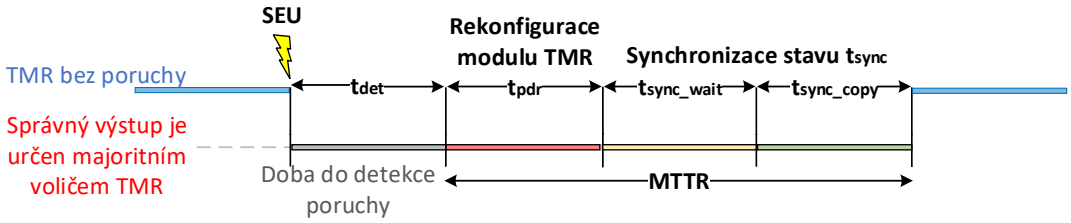
- *Obvodová režie implementace* – Obvodová režie udává poměr spotřebovaných zdrojů využitých k implementaci logiky synchronizace v daném obvodě FPGA k jeho celkovému počtu dostupných zdrojů.

- *Příkon* – Přidaný příkon způsobený implementací obvodů pro realizaci synchronizace stavu.

- *Redukce provozní frekvence* – Implementace dodatečné logiky pro synchronizaci stavu systému, zejména vytvoření propojení nebo úprava sekvenčních obvodů pro synchronizaci způsobuje pokles provozní frekvence. Provozní frekvence je ovlivněna přidáním dalších obvodových prvků do kritické cesty v návrhu FPGA.
- *Spolehlivost obvodů realizujících synchronizaci stavu* – Spolehlivost obvodů realizujících synchronizaci stavu systému je důležitá stejně jako spolehlivost zabezpečeného systému. Proto je nutné implementovat obvody zodpovědné za synchronizaci také jako systémy odolné proti poruchám.

6.5 Proces opravy stavu systému pomocí rekonfigurace a synchronizace

Proces opravy stavu systému z poruchy pomocí rekonfigurace a synchronizace je znázorněn na obrázku 6.4.



Obrázek 6.4: Proces opravy stavu systému z poruchy

Doba trvání procesu opravy stavu systému z poruchy je dána dle vzorce 6.1 součtem časů potřebných pro detekci poruchy, opravu poruchy a synchronizaci stavu systému.

$$t_{recov} = t_{det} + t_{pdr} + t_{sync} \tag{6.1}$$

- *Doba detekce poruchy* t_{det} – Doba detekce poruchy je proměnná a závisí hlavně na tom, jak rychle je systém za pomocí majoritních voličů (případně jiných detekčních mechanismů) schopen poruchu detekovat. Některé poruchy mohou zůstat skryté nebo mohou být aktivovány později.
- *Doba rekonfigurace* t_{pdr} – Doba rekonfigurace modulu PRM závisí na velikosti částečného bitstreamu pro daný PRM a rychlosti, s jakou je řadič rekonfigurace schopný provést rekonfiguraci.
- *Doba synchronizace* t_{sync} – Doba synchronizace může znamenat dobu, než stavový automat přejde do synchronizačního stavu nebo než se provede synchronizace stavu prostřednictvím kopie dat. Výpočet doby synchronizace může být proveden pomocí vzorce 6.2. Interval t_{sync_wait} označuje dobu čekání na přechod systému do synchronizačního stavu. Interval t_{sync_copy} označuje dobu kopie dat synchronizovaných objektů z jednoho modulu do druhého.

$$t_{sync} = t_{sync\_wait} + t_{sync\_copy} \tag{6.2}$$

6.6 Shrnutí

Aplikace metodiky pro návrh synchronizace stavu bude v dalším textu práce demonstrována na dvou typech systémů, systému řadiče sběrnice CAN řízeného stavovými automaty a systému mikrokontroleru s jádrem procesoru NEO430.

6.6.1 Předpoklady pro implementaci metody synchronizace dle navržené metodiky

Pro použití metodiky návrhu synchronizace stavu se předpokládá, že bude návrh systému splňovat následující kritéria:

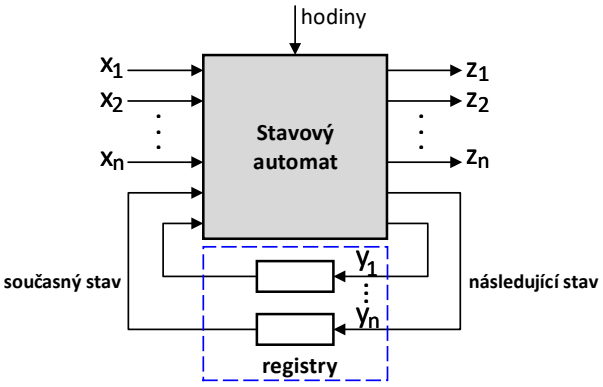
1. Systém je zabezpečen pomocí architektury TMR a každá z redundantních kopií má stejnou konfiguraci. (Např. procesor zabezpečený pomocí TMR má ve všech instancích stejný program a programová data.)
2. Všechny signály vedoucí do každého modulu TMR jsou synchronizovány mimo architekturu TMR.
3. Vstupní hodinový signál musí být přiveden do každého modulu TMR. Redundantní moduly musí pracovat stejně s přesností na jeden hodinový cyklus.
4. Vstupní nulovací signál reset musí být přiveden do každého modulu TMR. Redundantní moduly se musí restartovat ve shodném okamžiku.

Kapitola 7

Synchronizace stavu systému s dávkovým zpracováním dat

Systémy s dávkovým zpracováním dat jsou typicky aktivní pouze při zpracování vstupního bloku dat. Příkladem mohou být systémy provádějící periodické zpracování datových paketů síťové komunikace nebo obrazových informací. Tyto systémy se po dokončení své činnosti vrací do stavu nečinnosti, ve kterém čekají na příchod nových vstupních dat. Pro implementaci těchto systému v obvodech FPGA lze využít jednoho či více stavových automatů. V této kapitole je demonstrováno použití metodiky pro návrh synchronizace stavu systému řadiče komunikační sběrnice *CAN (Controller Area Network)* zabezpečeného pomocí rekonfigurovatelné architektury CG-TMR [89] [90] [93] [94].

Tato kapitola se blíže zabývá způsobem synchronizace stavu systému řízeného stavovými automaty. Schéma stavového automatu je znázorněno na obrázku 7.1. Systémy řízené stavovými automaty mají tu výhodu, že jsou v systému definovány všechny platné stavy, validní přechody mezi stavy a hodnoty výstupních signálu. Pří výběru synchronizačního stavu je nutné hledat stav splňující požadavky definované v kapitole 6.2.1, ve smyslu hledání synchronizační sekvence dle kapitoly 2.8.



Obrázek 7.1: Model stavového automatu

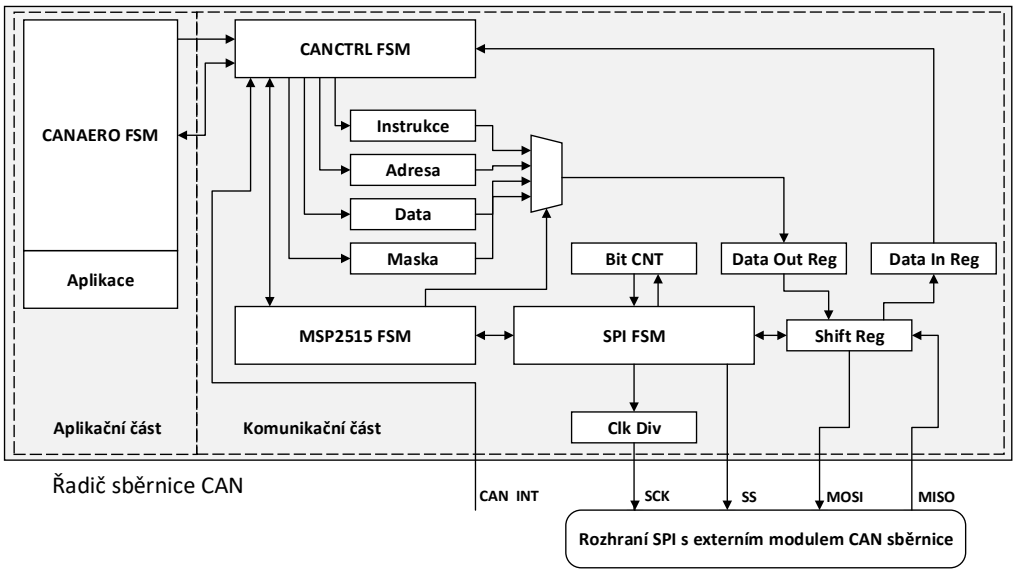
7.1 Návrh řadiče sběrnice CAN odolného proti poruchám

Řadič sběrnice CAN byl v rámci vlastního výzkumu implementovaný za účelem poskytnutí jednoduché a spolehlivé metody pro vzájemné propojení několika obvodů FPGA mezi sebou a pro propojení s jinými zařízeními připojenými ke sběrnici [89] [90].

Sběrnice CAN je sériová datová sběrnice, jež se stala široce rozšířeným průmyslovým standardem [79]. Sběrnice umožňuje komunikaci s maximální přenosovou rychlostí 1 MB/s. Při komunikaci zařízení připojených ke sběrnici může být každé zařízení tzv. master a řídit tak chování jiných zařízení, pokud je sběrnice v daný okamžik volná. Zprávy přenášené po sběrnici CAN neobsahují žádnou informaci o cílovém zařízení a jsou tedy přijímány všemi zařízeními. Nicméně každá zpráva je označena identifikátorem, který udává typ zprávy a její prioritu. Protokol sběrnice CAN definuje pouze fyzickou a linkovou vrstvu pro základní způsob komunikace. Základní funkci komunikačního protokolu sběrnice CAN lze rozšířit na úrovni aplikační vrstvy jako je tomu např. v případě protokolu *CANAerospace* [86]. Protokol CANAerospace rozděluje základní zprávu protokolu CAN na sémantickou a datovou část, díky čemuž lze definovat na úrovni aplikace další typy zpráv a implementovat specifické služby využívající přenášená data.

7.1.1 Řadič sběrnice CAN

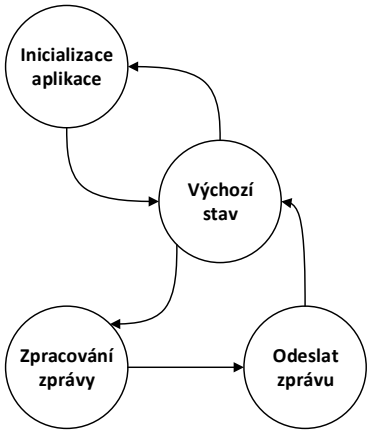
Řadič sběrnice CAN byl implementován v jazyce VHDL jako samostatná jednotka syntetizovatelná do FPGA. Pro samotné fyzické propojení přes sběrnici CAN je nutné použít malý elektronický modul, ve kterém jsou na desce plošných spojů umístěny obvody pro přijímání a vysíláni komunikačních rámců na fyzické úrovni sběrnice CAN. Jádrem elektronického modulu je obvod *MCP2515* [64] od firmy Microchip, se kterým je jednotka řadiče sběrnice CAN implementovaného uvnitř FPGA propojena pomocí sériového rozhraní *SPI (Serial Peripheral Interface)*. Prvotní návrh a implementace řadiče sběrnice CAN spolu s externím elektronickým modulem byly popsány v diplomové práci autora [89]. Architektura systému řadiče sběrnice CAN je znázorňena na obrázku 7.2.



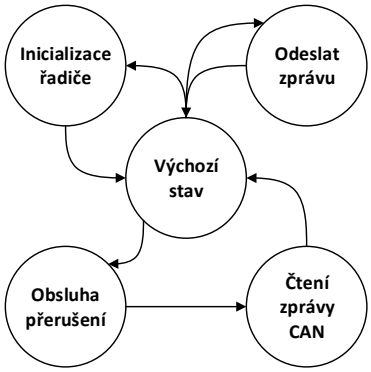
Obrázek 7.2: Architektura řídicího systému sběrnice CAN

Implementovaný systém se skládá z aplikační a komunikační části. V rámci komunikační části je systém řízen pomocí jednotky *CANCTRL*. Tato jednotka implementuje stavový automat, který řídí komunikaci s obvodem MCP2515 umístěným na externím elektronickém modulu. Komunikace s obvodem MCP2515 se provádí přes rozhraní SPI. Jednotka CANCTRL také zpracovává příchozí žádosti o obsluhu přerušení a komunikuje s aplikační částí systému. Na schématu 7.2 je vidět několik vzájemně komunikujících stavových automatů, které implementují logiku řadiče sběrnice CAN na několika úrovních: na úrovni komunikačního protokolu sběrnice CAN, na úrovni komunikace s obvodem MCP2515 a na úrovni zpracování komunikaci přes rozhraní SPI. Implementované stavové automaty jsou znázorněny na obrázku 7.3.

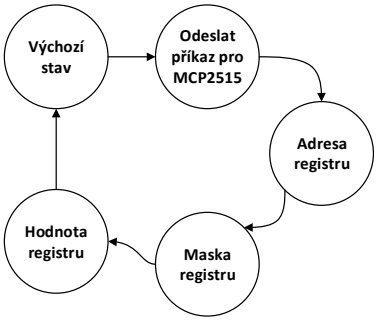
Aplikační část systému je tvořena komponentou *CANAERO*, která implementuje specifické funkce pro řízení komunikace po sběrnici CAN a pro zpracování přenášených komunikačních rámců. *CANAERO* využívá aplikační protokol CANAerospace [86], který umožňuje přenášet v datových rámcích spolu s daty také jejich datový typ, definovat prioritu rámců a bližší význam. Pro experimenty byla do aplikační vrstvy implementována podpora pro distribuované matematické výpočty mezi systémy připojenými na sběrnici CAN. V systému byla pro testovací účely implementována aplikace distribuovaného kalkulátoru, ve které byly jednotlivé matematické operace realizovány v podobě služeb implementovaných za pomocí protokolu CANAerospace.



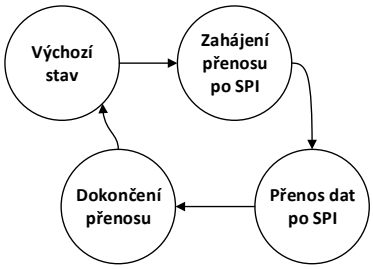
(a) Stavový automat CANAERO



(b) Stavový automat CANCTRL



(c) Stavový automat MCP2515

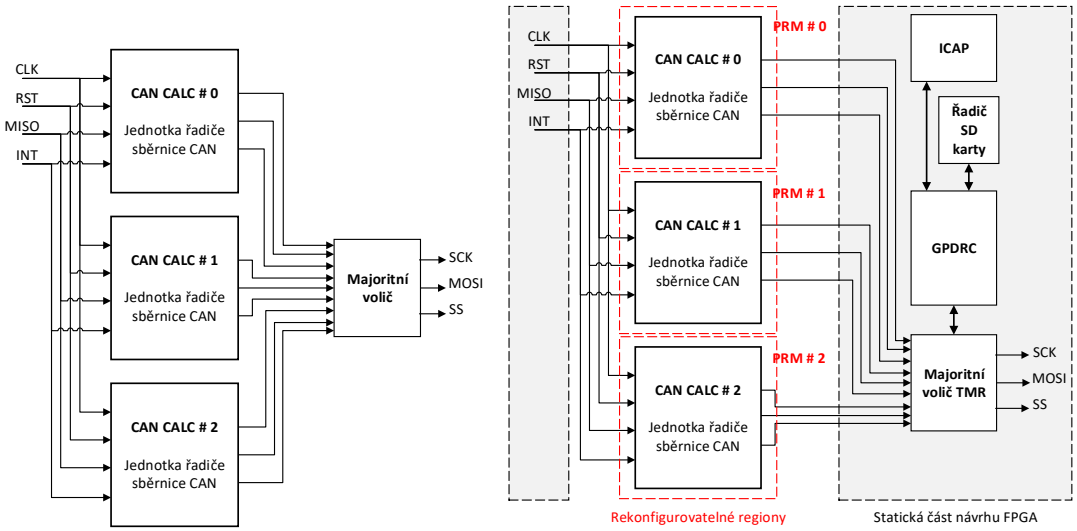


(d) Stavový automat SPIMaster

Obrázek 7.3: Zjednodušená schémata stavových automatů v řadiči sběrnice CAN

7.1.2 Rekonfigurovatelná architektura odolná proti poruchám

Řídicí systém byl nejdříve zabezpečen pomocí architektury CG-TMR s cílem zvýšení celkové spolehlivosti, jak je znázorněno na obrázku 7.4a. Jednoduchý volič byl připojen na výstupy ztrojených instancí řadiče pro maskování poruch při komunikaci po rozhraní SPI s externím modulem.



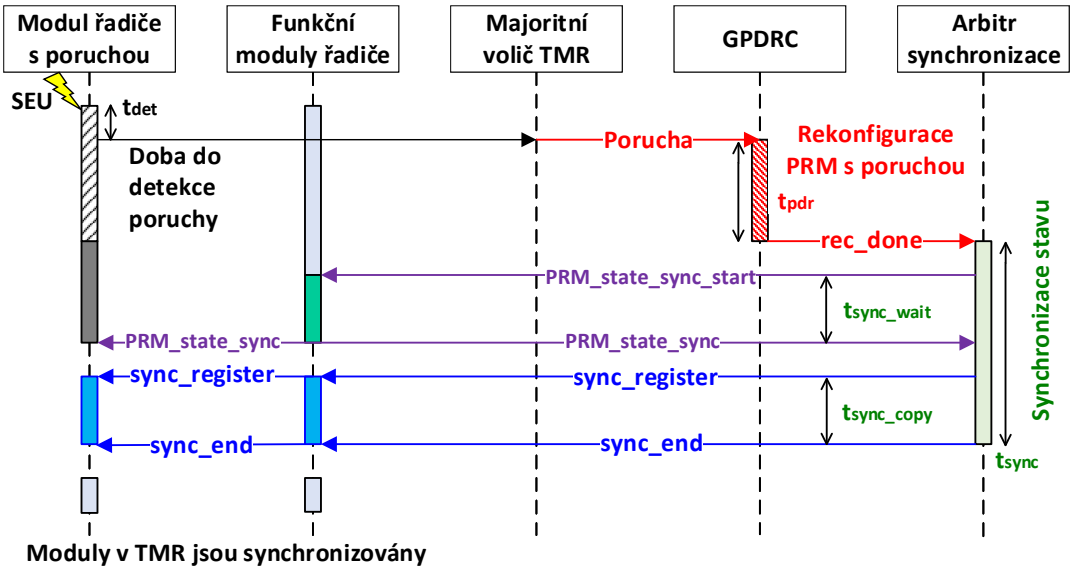
(a) Statický návrh TMR (b) Rekonfigurovatelná architektura

Obrázek 7.4: Schéma řadiče sběrnice CAN zabezpečeného pomocí TMR

Na implementaci systému s architekturou CG-TMR, znázorněného na obrázku 7.4a, navázal návrh a implementace rekonfigurovatelného systému s cílem opravit některou z kopií chráněného řídicího systému v případě, že je detekována porucha. Rekonfigurovatelnou architekturu celého systému znázorňuje obrázek 7.4b.

Systém je rozdělen v konfigurační paměti na statickou a dynamickou část. Dynamická část je rozdělena na rekonfigurovatelné regiony, do kterých jsou umístěny jednotlivé redundantní instance systému řadiče sběrnice CAN. Statická část integruje řadič rekonfigurace GPDRC, komponentu rozhraní ICAP a řadič pro kartu SD. Na kartě SD jsou uloženy předem připravené konfigurační soubory bitstream. Dále je zde umístěn majoritní volič TMR, provádějící maskování a detekci poruch v kopiích chráněného řídicího systému sběrnice CAN. V případě, že v některé kopii řídicího systému je detekována porucha, je provedena rekonfigurace odpovídající oblasti PRM.

Průběh procesu opravy stavu řadiče sběrnice CAN pomocí rekonfigurace a synchronizace stavu je znázorněn na obrázku 7.5. Během rekonfigurace redundantního modulu, ve kterém byla detekována porucha, je systém stále v provozu. Po dokončení rekonfigurace je nutné provést synchronizaci opraveného modulu s ostatními. Proces synchronizace byl implementován podle popsané metodiky v kapitole 6.



Obrázek 7.5: Časový diagram opravy a synchronizace řadiče sběrnice CAN z poruchy

7.2 Návrh techniky synchronizace stavu řadiče s využitím navržené metodiky

V první řadě bylo nutné analyzovat celý řídicí systém. Jak již bylo zmíněno v kapitole 7.1.1, systém řadiče sběrnice CAN je řízen pomocí čtyř vzájemně propojených stavových automatů rozdělených do aplikační a komunikační vrstvy. Metoda synchronizace stavu pro rekonfigurovatelné moduly řadiče sběrnice CAN, zabezpečeného pomocí TMR, byla vzhledem k funkčnímu rozdělení návrhu systému rozdělena do dvou fází:

- Synchronizace komunikační vrstvy – Komunikační vrstva řadiče je řízena jednotkou *CANCTRL*, která zpracovává příchozí požadavky přerušení nebo řídí komunikaci s připojeným obvodem MCP2515. V případě, že je jednotka neaktivní, přechází vnitřní automat do klidového stavu *IDLE*. Protože je tento stav vždy dosažen a prováděné operace jsou relativně krátké, byl tento stav zvolen jako synchronizační. Synchronizace stavových automatů komunikační vrstvy se tedy provede přepnutím synchronizovaného systému do tohoto stavu v okamžiku, kdy jej dosáhne také referenční systém.

- Synchronizace stavu aplikace – Aplikační vrstva řadiče je řízena jednotkou *CANAERO*. Tato jednotka spouští inicializaci systému obvodu MCP2515, zpracovává zprávy ve formátu *CANAerospace* a provádí aplikační výpočty dle dat přijatých ve zprávě. Jednotka obsahuje stavové registry, které obsahují data uložená během provozu systému a provádění matematických operací. Stav systému v tomto případě záleží i na předchozích výpočtech systému a je tedy nutné provést synchronizaci stavu založenou na kopii všech stavových registrů z referenčního systému do synchronizovaného.

7.2.1 Implementace logických obvodů pro synchronizaci stavu

Návrh logických obvodů realizujících synchronizaci stavu v rekonfigurovatelné architektuře řadiče sběrnice CAN byl založen na principech popsaných v kapitole 6.3. Arbitr synchronizace je umístěn do statické oblasti v konfigurační paměti FPGA. Řadič synchronizace je implementován do každého PRM spolu s kopií řídicího systému sběrnice CAN. Řadič synchronizace je propojen s GPDRC, který po dokončení rekonfigurace dané oblasti PRM povolí pomocí signálu *rec_done* provedení synchronizace. Současně GPDRC předá pomocí signálu *PRM_index* informace o oblasti PRM, která byl rekonfigurována a vyžaduje synchronizaci. Synchronizační arbitr na základě hodnoty *PRM_index* identifikuje, který systém v PRM má být synchronizován a který lze použít jako referenční.

7.2.2 Synchronizace stavu komunikační vrstvy řadiče

Princip synchronizace sekvenčních automatů v komunikační vrstvě řadiče je založen na čekání na okamžik, kdy sekvenční automaty v referenčním systému přejdou do specifického stavu, a nastavení stavu synchronizovaného systému do téhož stavu.

Synchronizace v architektuře CG-TMR je provedena následovně. Během obnovy stavu systému je rekonfigurovaný modul pozastaven, kdy čeká na synchronizaci. Ostatní moduly stále pracují. Synchronizace je implementována na úrovni jednotky CANCTRL. Tato jednotka je připojená k řadiči synchronizace. Začátek synchronizace je indikován signálem *PRM_state_sync_start*. Řadič synchronizace v referenčním modulu začne čekat na přechod jednotky CANCTRL do stavu IDLE. Ve chvíli přechodu jednotky CANCTRL do tohoto stavu, řadič synchronizace aktivuje signál *PRM_state_sync* připojený k arbitru synchronizace. Arbitr synchronizace pozastaví funkci všech redundantních modulů a přejde k synchronizaci aplikační vrstvy.

7.2.3 Synchronizace stavu aplikační vrstvy řadiče

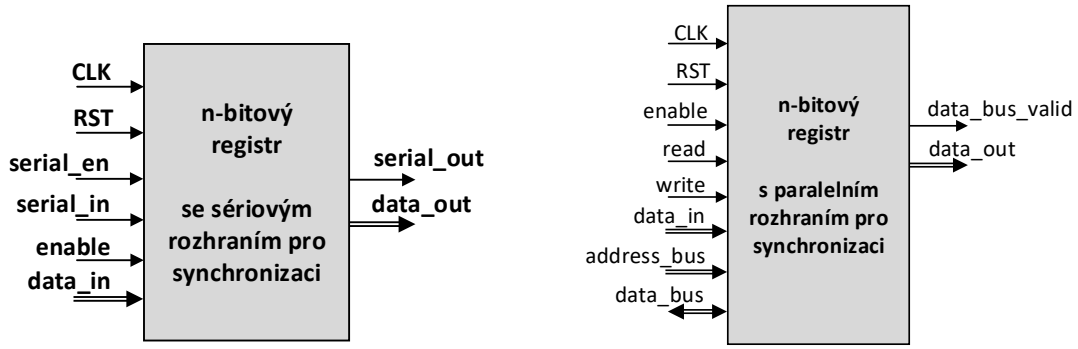
Jednotka CANAERO disponuje několika datovými registry, které uchovávají stav aplikace. Příkladem mohou být registry, které uchovávají výsledky matematických výpočtů prováděných na základě příkazů přijatých přes sběrnici CAN.

Pro synchronizaci těchto registrů bylo nutné navrhnout specifické synchronizační propojení umožňující jejich přenos z referenčního do synchronizovaného modulu a modifikovat implementaci registrů pro přístup během synchronizace. Schémata upravených registrů jsou znázorněny na obrázcích 7.6a a 7.6b.

Synchronizace aplikačních registrů pomocí sériového propojení

Synchronizace stavu pomocí sériového propojení registrů je založená na principu vytvoření řetězce všech registrů do jednoho posuvného registrů, který má jeden sériový vstup *serial_in* a jeden sériový výstup *serial_out*. Všechny synchronizované registry byly upraveny podle návrhu zobrazeného na schématu 7.6a. Synchronizační logika posuvných registrů je funkční pouze při aktivaci vstupu *serial_en*. Pro normální funkci registr umožňuje paralelní vstup a paralelní výstup pomocí signálů *data_in* a *data_out*.

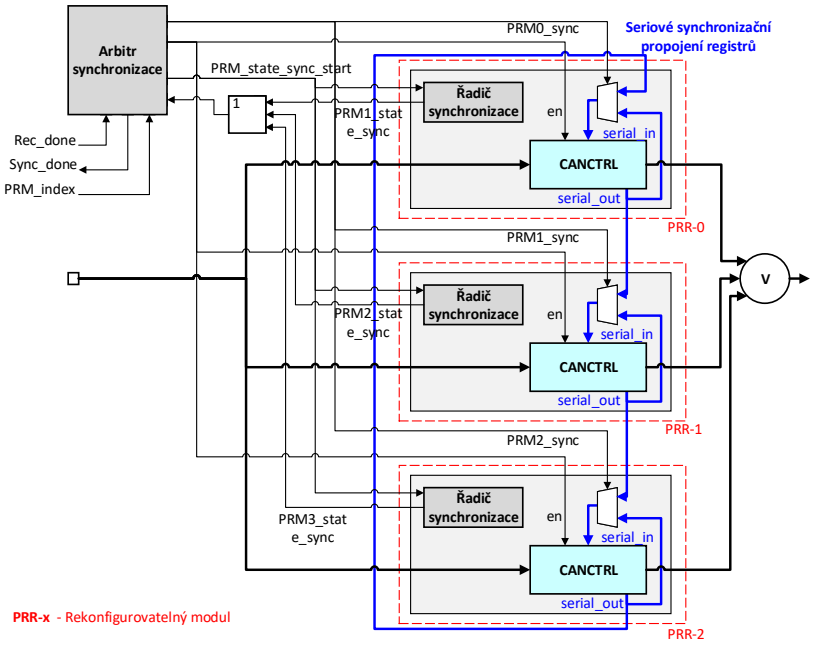
Pro synchronizaci rekonfigurovaného modulu je na jeho vstup *serial_in* přiveden výstup *serial_out* z referenčního redundantního modulu. Po provedení potřebného počtu synchronizačních cyklů v rámci posuvného registru je dokončena synchronizace všech registrů



(a) Registr pro sériovou synchronizaci (b) Registr pro paralelní synchronizaci

Obrázek 7.6: Ručně upravené regitry pro synchronizaci na úrovni RTL

v synchronizovaném modulu. Při synchronizaci je výstup *serial_out* bezporuchových modulů přiveden zpět na vstup *serial_in* stejného modulu.



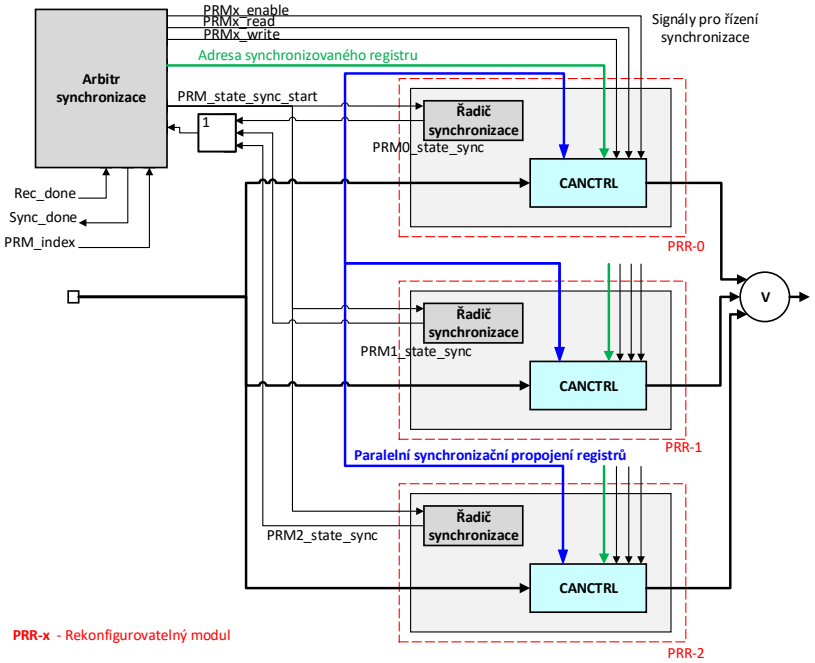
Obrázek 7.7: Schéma synchronizace stavu pomocí sériového propojení

Synchronizace aplikačních registrů pomocí paralelního propojení

Druhá varianta implementované metody synchronizace stavu registrů je založená na paralelním propojení pro přenos hodnot registrů mezi redundantními moduly. Jednotlivé registry jsou zpřístupněny pomocí adresové a datové sběrnice. Všechny synchronizované registry byly upraveny podle návrhu zobrazeného na schématu 7.6b. Každý implementovaný registr má přiřazenu specifickou adresu, díky které je identifikován na adresové sběrnici. Přístup

k registru se aktivuje v případě, že je na adresové sběrnici *address_bus* vystavena odpovídající adresa cílového registru a signál *read* nebo *write* je aktivní.

Schéma paralelního propojení pro synchronizaci stavu registrů aplikační části řadiče sběrnice CAN je znázorněno na obrázku 7.8. Princip synchronizace stavu přes paralelní propojení využívá mechanismus adresování jednotlivých registrů pomocí adresové sběrnice, signálů *PRM_write* a *PRM_read* pro zpřístupnění registrů během synchronizace.



Obrázek 7.8: Schéma synchronizace stavu pomocí paralelní sběrnice

7.3 Experimentální a implementační výsledky

Experimenty pro ověření procesu opravy a zotavení stavu systému zabezpečeného řadiče sběrnice CAN po poruše byly provedeny na vývojové desce ML506 s obvodem FPGA Xilinx Virtex5. Během experimentů byly poruchy typu SEU náhodně injektovány do rekonfigurovatelných oblasti PRM v konfigurační paměti FPGA s moduly TMR zabezpečeného systému řadiče sběrnice CAN.

V tabulce 7.1 jsou shrnuty délky trvání různých operací, které řadič sběrnice CAN a implementovaná aplikace provádějí během svého provozu. Implementovaný návrh systému v FPGA běžel na hodinové frekvenci 100 MHz. Rychlost komunikace po sběrnici CAN byla limitována implementací systému umožňující komunikaci přes rozhraní SPI s maximální frekvencí 1.25 MHz. Řadič sběrnice CAN provádí několik operací během svého provozu. Počáteční inicializace obvodu MCP2515, který je připojen na externím elektronickém modulu k FPGA přes rozhraní SPI, trvá 600.7 µs. Příjem zprávy po sběrnici CAN je indikován požadavkem na přerušení od obvodu MCP2515, který je zpracován v jednotce CANCTRL. Zpracování požadavku na přerušení trvá 45.7 µs. Následný přenos zprávy CANAerospsace, která je uložena v komunikačním rámci protokolu CAN o velikosti 13 B, trvá 275.8 µs. Přičemž přenos jedné slabiky (1B) přes rozhraní SPI má délku trvání 6.8 µs.

Operace	Délka trvání
Inicializace obvodu MCP2515	600.7 µs
Zpracování požadavku na přerušení	45.7 µs
Přenos zprávy CANAerospace	275.8 µs
Přenos 1B přes rozhraní SPI	6.8 µs

Tabulka 7.1: Délka trvání operací prováděných v řadiči sběrnice CAN

V tabulce 7.2 jsou shrnuty délky trvání operací rekonfigurace a synchronizace stavu pro implementaci rekonfigurovatelné architektury zabezpečeného řadiče sběrnice CAN. Rekonfigurace redundantního modulu architektury TMR, která zabezpečuje řadič sběrnice CAN, je provedena ihned v případě detekce poruchy pomocí majoritního voliče TMR. Rekonfigurace je provedena za běhu ostatních redundantních modulů, které pracují bez poruchy. Rekonfigurace jednoho PRM, kterému odpovídá částečný konfigurační bitstream o velikosti 51.5 kB, trvá 2730 µs.

Operace	Délka trvání
Rekonfigurace PRM	2730 µs
Synchronizace komunikační vrsty	275.8 µs
Sériová synchronizace aplikační vrstvy	0.82 µs
Paralelní synchronizace aplikační vrsty	0.11 µs

Tabulka 7.2: Délka rekonfigurace a synchronizace stavu řadiče sběrnice CAN

Po dokončení rekonfigurace je aktivována logika pro synchronizaci stavu rekonfigurovaného modulu s ostatními redundantními moduly systému. Délka synchronizace stavu sekvenčních automatů v komunikační vrstvě řadiče sběrnice CAN je dána čekáním na návrat do výchozího stavu automatů. V nejhorším případě by toto čekání trvalo 600.7 µs, během inicializace řadiče a 275.8 µs během přenosu příchozí zprávy CANAerospace.

Synchronizace aplikační vrstvy je provedena ihned po dokončení synchronizace komunikační vrstvy. Délka trvání synchronizace stavu registrů pomocí sériového propojení trvá 0.82µs a pomocí implementace paralelního propojení registrů pouze 0.11 µs. Časová náročnost synchronizace registrů je přímo úměrná jejich počtu. V implementované aplikaci řadiče sběrnice CAN bylo implementováno pouze 5 registrů, které celkově uchovávali 75 bitů. Rozdíl mezi časovou náročností obou variant synchronizace je značný, nicméně odpovídá typu zvoleného propojení a implementační režii synchronizační logiky.

V tabulce 7.3 je uvedeno porovnání spotřebovaných zdrojů FPGA pro implementaci řadiče sběrnice CAN, komponent potřebných pro rekonfiguraci FPGA a obvodů realizujících synchronizaci stavu v sériové a paralelní variantě propojení.

7.4 Shrnutí

V této kapitole byly popsány výzkumné aktivity, jejichž snahou bylo navrhnout a implementovat mechanismus opravy systému implementovaného v FPGA po vzniku poruchy typu SEU a techniku synchronizace stavu vedoucí k zotavení stavu systému CG-TMR. Brzká oprava architektury TMR v případě poruchy jednoho z modulů, je nutná pro zachování vysoké dostupnosti systému. Na implementaci zabezpečeného systému řadiče sběrnice CAN byl předveden způsob opravy poruchy v FPGA pomocí částečné dynamické rekonfigurace a

Varianty systému	Sériové propojení		Paralelní propojení	
Virtex5 XC5VSX50T Implementované komponenty	Počet registrů	Počet LUT	Počet registrů	Počet LUT
Řadič sběrnice CAN (3x)	1284	1917	1283	2456
+ SPI Master	49	49	49	49
+ MCP2515	45	44	45	45
+ CANCTRL	254	227	250	226
+ CANAERO	80	285	88	384
Majoritní volič TMR	0	6	0	9
Jednotka GPDRC	151	256	151	256
Řadič karty SD	211	479	211	479
Řadič synchronizace	0	1	0	1
Arbitr synchronizace	24	36	8	17
Ostatní logika	38	16	50	48
Celkem	1708	2711	1704	3265

Tabulka 7.3: Přehled spotřebovaných zdrojů FPGA implementací řadiče sběrnice CAN

specifická strategie synchronizace stavu. Podle vytvořené metodiky pro návrh metod synchronizace stavu, která byla popsána v kapitole 6, byly implementovány tři metody pro synchronizaci stavu vnitřních komponent řadiče sběrnice CAN.

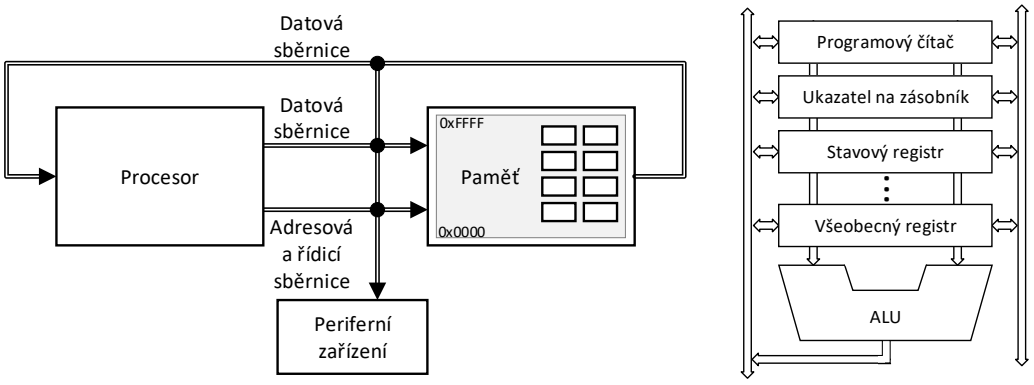
Kapitola 8

Synchronizace operačního stavu procesorového systému

Tato kapitola demonstruje použití metodiky pro návrh synchronizace stavu na systému řízeném jádrem procesoru NEO430, který je zabezpečen pomocí rekonfigurovatelné architektury CG-TMR. Systém, který je řízen jádrem procesoru, je daleko složitější než systém řízený pouze pomocí jednoho nebo více stavových automatů. Z tohoto důvodu bylo pro ověření správné funkce implementované logiky realizující zabezpečení systému, rekonfiguraci a synchronizaci stavu systému, vytvořeno testovací prostředí PDR Framework. Experimenty s využitím tohoto prostředí jsou popsány v podkapitole 8.5.

8.1 Synchronizace stavu pro systém řízený procesorem

Tato kapitola se blíže zabývá způsobem synchronizace stavu systému řízeného procesorem. V následujícím textu je uvažován model systému v podobě mikrokontroleru implementovaného uvnitř FPGA. Schéma hypotetického mikrokontroleru je znázorněno na obrázku 8.1a. Mikrokontroler se zjednodušeně skládá z řídicího jádra procesoru, vnitřní operační paměti (pro program a data) a periferních zařízení. Periferní zařízení mohou sloužit pro komunikaci s vnějším prostředím, pro zpracování vstupních signálů a generování výstupních signálů.



(a) (b)

Obrázek 8.1: Model mikrokontroleru a aritmeticko-logické jednotky

8.1.1 Stav procesoru

Architektura většiny procesorů je z pohledu definice stavu procesoru podobná, pomineme-li existující rozdíly dané počtem jader procesoru, instrukční sadou, velikostí zřetězené linky vykonávání instrukcí apod. Procesory se mohou lišit také v principu přístupu k paměti. Procesory s harvardskou architekturou mají oddělenou programovou a datovou paměť. Alternativou je Von Neumannova architektura, u které má procesor společnou paměť pro program i data. Výkonnější a složitější procesory implementují dodatečné vyrovnávací paměti, které umožňují optimalizaci přístupu do paměti.

Většina procesorů se skládá z aritmeticko-logické jednotky ALU, jednotky generátoru adres, obvodů pro čtení a dekódování instrukcí. Tyto funkční jednotky implementující vnitřní architekturu procesoru mohou obsahovat registry, které slouží k uchování mezivýsledků prováděných operací a které nejsou z vnějšku viditelné. Naopak, každý procesor obsahuje sadu různých registrů, které definují stav procesoru z hlediska programátora.

Na základě určitě podobnosti mezi procesory lze definovat následující synchronizační objekty, které definují stav procesoru a který je nutné synchronizovat:

- *Programový čítač PC (Program Counter)* – Programový čítač obsahuje adresu další instrukce, kterou procesor vykoná. Programový čítač je automaticky inkrementován v případě, že procesor vykonává lineární sekvenci instrukci. V případě skoků je programový čítač nastaven na cílovou adresu.
- *Ukazatel na zásobník SP (Stack Pointer)* – Ukazatel na zásobník je registr, který ukazuje na poslední hodnotu vloženou do zásobníku.
- *Řídicí a stavový registr* – Nejdůležitější registr z programátorského hlediska. Stavový registr většinou obsahuje příznaky výsledku poslední aritmetické operace, příznaky pro povolení přerušení a dalších řídicích funkcí procesoru.
- *Všeobecné registry* – Procesor má většinou k dispozici několik všeobecných registrů, které jsou využity pro uchování proměnných v programu. Některé všeobecné registry jsou často využity také pro uložení argumentů funkce nebo návratové hodnoty funkce.
- *Programový zásobník* – Programový zásobník může být součástí datové oblasti operační paměti nebo být implementován pomocí dedikované paměti. Např. procesor Xilinx PicoBlaze disponuje hardwarovým zásobníkem pro uložení až 31 návratových adres při volání programových funkcí nebo rutin obsluhy přerušení [107].
- *Data uložená v operační paměti* – V datové paměti (resp. v datovém regionu operační paměti, pokud se jedná o Von Neumannovu architekturu) jsou uloženy globální programové proměnné, za běhu dynamicky alokovaná paměť (tzv. hromada resp. anglicky heap), případně programový zásobník.

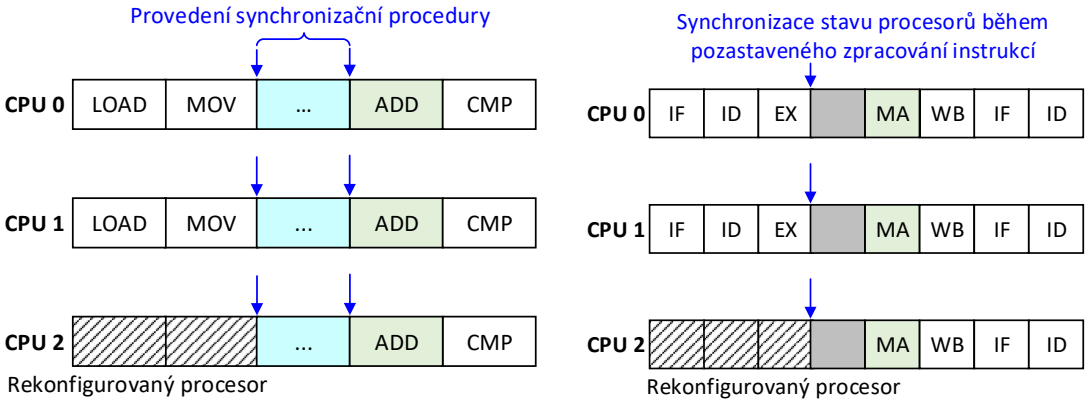
8.1.2 Synchronizace během zpracování instrukcí procesoru

Synchronizace provádění programu v procesoru může být implementována na úrovni provádění instrukcí programu nebo na úrovni provádění jednotlivých stupňů instrukční pipeline.

Synchronizace na úrovni instrukcí je znázorněna na obrázku 8.2a. V takovém případě jsou procesory synchronizovány programově. Po dokončení synchronizační procedury ukazuje registr *PC* ve všech procesorech na stejnou instrukci. Případně při implementaci synchronizace v obslužné rutině přerušení je *PC* synchronně nastaven při návratu z přerušení.

Jestliže procesor provádí zřetězené zpracování instrukcí, pak by měla být synchronizační procedura implementována s ohledem na zamezení vzniku datových konfliktů mezi instrukcemi a synchronizovanými objekty procesoru. Pro komplexnější procesory to může znamenat nutnost optimalizace pořadí zpracování instrukcí.

Chceme-li provést synchronizaci procesoru bez podpory programu pouze s využitím obvodových prostředků, pak je nutné provést synchronizaci procesoru na úrovni provádění jednotlivých stupňů instrukční pipeline. Tento způsob synchronizace je znázorněn na obrázku 8.2. Pro provedení tohoto způsobu synchronizace musí mít všechny synchronizované objekty implementované synchronizační rozhraní a jednotlivé procesory musí být vzájemně propojeny pomocí sériového nebo paralelního propojení. Pro zachování integrity a konzistence stavu procesorů musí být zpracování instrukcí programu v procesorech po dobu synchronizace zastaveno. Chod obvodů provádějících synchronizaci stavu může být řízen doplňujícím hodinovým signálem, aktivním pouze během synchronizace.



(a) Synchronizace na úrovni instrukcí (b) Synchronizace na úrovni instrukční pipeline

Obrázek 8.2: Úrovně synchronizace provádění procesoru

8.1.3 Návrh metod synchronizace stavu procesoru

Pro synchronizaci stavu procesoru lze využít metody souhrnně popsané v kapitole 6.2.3. Vhodnost použití těchto metod je porovnána v tabulce 8.1.

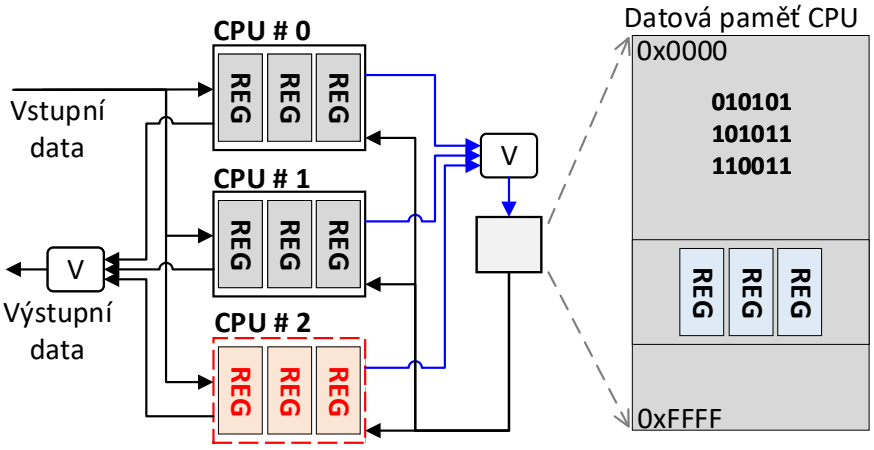
Metoda synchronizace	Úprava arch. procesoru	Režie HW	Režie SW
Synchronizační restart	Ne	Malá	Malá
Přenesení stavu	Není vhodné	–	–
Paralelní propojení	Velká (propojení registrů)	Velká	Žádná
Sériové propojení	Velká (propojení registrů)	Velká	Žádná
Sdílená paměť	Malá (propojení CPU s paměti)	Malá	Větší
Synchronizační sběrnice	Ne (využití V/V rozhraní)	Malá	Velká

Tabulka 8.1: Porovnání implementace synchronizace stavu v procesoru

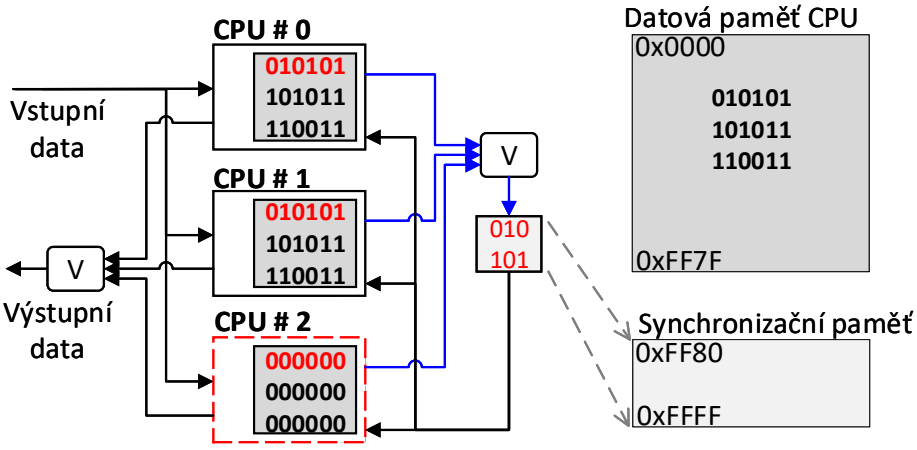
Návrh propojení procesorů na úrovni redundantních modulů

Způsob návrhu a implementace synchronizace s využitím paralelního a sériového propojení redundantních modulů byl ukázán na řadiči sběrnice CAN v kapitole 7.2. Návrh synchronizace s využitím propojení na úrovni registrů by pro procesor znamenal velký zásah do jeho architektury a způsobil by značnou implementační režii. Provedení synchronizace na úrovni registrů by také vyžadovalo pozastavení běhu procesoru pro zamezení konfliktů na úrovni instrukční pipeline, jak je znázorněno na obrázku 8.2b.

Pokud se zaměříme na metody synchronizace stavu procesoru provádějící kopii a přenos synchronizačních objektů z bezporuchového procesoru do procesoru, který byl rekonfigurován, pak je nutno při návrhu techniky synchronizace jednotlivých typů synchronizačních objektů zvážit způsob propojení procesorů v redundantních modulech TMR. Metody synchronizace využívající sdílenou paměť jsou znázorněny na obrázcích 8.3 a 8.4.

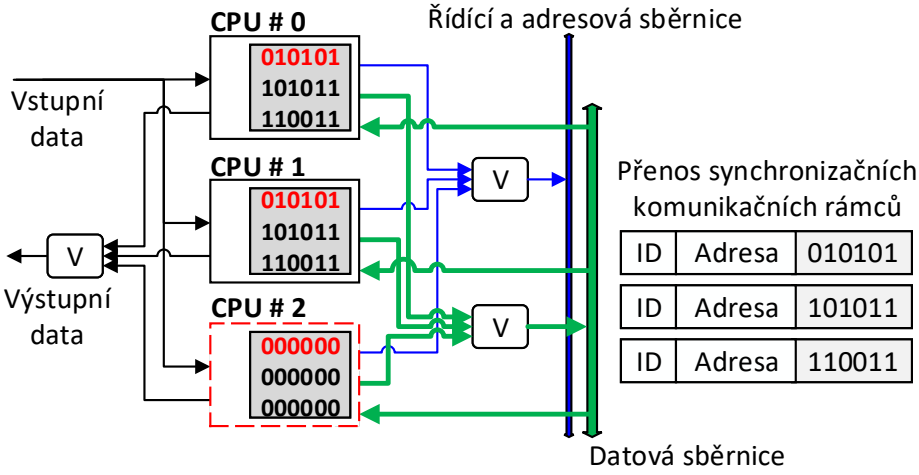


Obrázek 8.3: Synchronizace procesoru se sdílenou datovou pamětí



Obrázek 8.4: Synchronizace procesoru se sdílenou synchronizační pamětí

Metoda synchronizace využívající synchronizační sběrnici je znázorněna na obrázku 8.5. Tento způsob synchronizace vyžaduje použití V/V rozhraní procesoru ke komunikační sběrnici. Příkladem může být využití sběrnice Wishbone, CAN nebo jednoduché sériové komunikace. Tato metoda synchronizace vyžaduje implementaci programových funkcí pro řízení komunikace a zpracování komunikačních rámců přenášených po sběrnici. Procesor, který má být synchronizovaný, musí být také opětovně inicializován a připraven přijmout a zpracovat příchozí data synchronizačních objektů. Další nevýhodou tohoto přístupu je nutnost synchronizace obsahu datové paměti, kterou má každý procesor vlastní. Synchronizace vnitřních registrů procesoru může být velmi obtížná, protože pro jejich přenos po sběrnici musí procesor provést množství dalších instrukcí. Z tohoto důvodu se tento způsob synchronizace hodí spíše pro synchronizaci procesorů na vyšší úrovni abstrakce stavu procesorů, případně by měl být kombinován s jinou metodou synchronizace.



Obrázek 8.5: Synchronizace procesoru pomocí sběrnice

Synchronizace registrů procesoru

Pro synchronizaci všech registrů procesoru (programového čítače, ukazatele na zásobník, řídicího a stavového registru a všeobecných registrů) lze implementovat propojení na úrovni registrů nebo na úrovni modulů. Jsou-li všechny registry přístupné programově, tzn. je-li možné programově vyčíst a nastavit hodnotu celého registru, pak je optimálním řešením využití propojení na úrovni modulů. Implementace synchronizace na úrovni registrů vyžaduje úpravu architektury procesoru, což způsobuje velkou implementační režii.

Synchronizace programového zásobníku

Programový zásobník v procesoru může být součásti datové paměti RAM nebo implementován jako dedikovaná paměť. Synchronizace programového zásobníku, který je součástí datové paměti RAM, může být provedena v rámci synchronizace stavu paměti, popsané v následující podkapitole. Některé procesory mohou mít implementován programový zásobník ve formě speciální paměti. Příkladem je hardwarový zásobník procesoru Xilinx PicoBlaze. Způsob synchronizace hardwarového zásobníku byl popsán v [54].

Synchronizace stavu pamětí

Obsah instrukční paměti synchronizovaného procesoru se za běhu nemění a není nutné jej synchronizovat. (Pomineme-li speciální případy, kdy je z nějakého důvodu instrukční paměť modifikována.) Pro synchronizaci datové paměti lze uvažovat následující případy:

a) synchronizace pomocí majoritně voleného zápisu do sdílené paměti a zpětného čtení,

b) synchronizace jednotek datové paměti procesorů mezi redundantními moduly.

Pokud je datová paměť implementována jako sdílená mezi všemi procesory, pak není nutná její synchronizace. Sdílením paměti mezi procesory lze obejít nutnost synchronizace uložených dat a s tím spojenou časovou a implementační režii. Schéma synchronizace procesoru pomocí sdílené datové paměti je znázorněno na obrázku 8.3. Jestliže nelze implementovat celou datovou paměť jako sdílenou, pak může být kompromisem implementace malé sdílené synchronizační paměti. Tento způsob synchronizace je znázorněn na obrázku 8.4. Pro přístup do synchronizační paměti je nutné rozšířit nebo modifikovat paměťovou mapu procesoru. Synchronizace datové paměti s využitím synchronizační paměti je provedena v rámci synchronizační procedury synchronním zápisem a zpětným vyčtením všech zapsaných dat. V případě implementaci datové paměti procesoru v každém modulu TMR je nutné provést synchronizaci pomocí synchronizačního propojení mezi procesory. Po dokončení rekonfigurace procesoru je datová paměť znovu inicializována a je tedy nutné přenést hodnoty všech globálních proměnných, obsah zásobníku a dynamicky alokované pamětí.

Synchronizace periferních zařízení procesoru

Registry uvnitř periferních zařízení mohou být synchronizovány podobně jako registry uvnitř procesoru. Alternativou je provést pouze synchronizaci procesoru a periferní zařízení synchronizovat společným restartem. Restart funkce periferních zařízení vyžaduje, aby byly dokončeny všechny vstupní a výstupní operace. Následně musí provést program procesoru opětovnou konfiguraci všech použitých zařízení.

Synchronizace cache pamětí

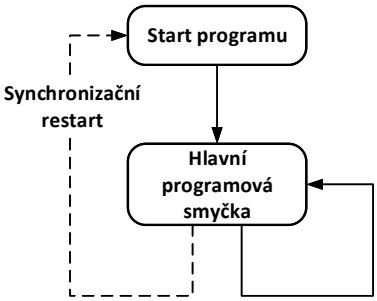
Jednoduchým způsobem pro synchronizaci paměti cache je programově řízené zneplatnění obsahu vyrovnávací paměti provedené během synchronizace instancí procesoru. Po dokončení synchronizace si procesory obnoví obsah paměti cache již synchronně. Tato disertační práce se dále synchronizací paměti cache procesoru nezabývá. Využití pamětí cache pro zrychlení přístupu procesoru k datům nebo instrukcí se používá u složitějších procesorů než je procesor NEO430 použitý v této disertační práci pro praktické experimenty.

8.1.4 Programová synchronizace stavu procesorů

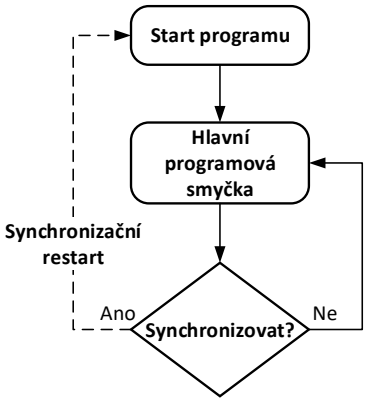
Základní způsob provádění programu při synchronizaci procesorů pomocí synchronizačního restartu je znázorněn na vývojovém diagramu na obrázku 8.6a. Procesory v CG-TMR vykonávají hlavní programovou smyčku do té doby, dokud není proveden synchronizační restart. Synchronizační restart procesorů může být proveden ihned po dokončení rekonfigurace modulu s procesorem, ve kterém byla detekována porucha. Tento způsob implementace synchronizace může být vylepšen přidáním periodické kontroly požadavku k synchronizaci, která umožní provést přechod programu do bezpečného stavu a dokončit prováděnou úlohu.

Požadavek na synchronizaci může být indikován např. digitálním signálem nebo nastavením příznaku ve sdílené paměti. Vývojový diagram programu, který zohledňuje současný stav provádění programových úloh je znázorněn na obrázku 8.6b.

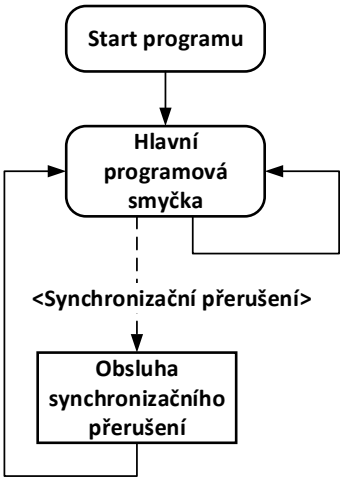
Požadavek na synchronizaci může být po dokončení rekonfigurace indikován procesorům pomocí externího přerušení. V takovém případě je chod hlavní programové smyčky v procesorech přerušen a synchronizace procesorů je provedena v obslužné rutině přerušení. Tento způsob synchronizace předpokládá, že všechny synchronizační objekty jsou přístupné programově přes sdílenou paměť nebo synchronizační sběrnici. Celý proces synchronizace procesorů může být proveden v rámci obsluhy přerušení procesoru. Vývojový diagram programu se synchronizací implementovanou v obsluze přerušení je znázorněn na obrázku 8.6c. Na této myšlence může být založen způsob synchronizace procesorů, znázorněný na vývojovém diagramu na obrázku 8.6d. Procesory jsou před synchronizací přivedeny do klidového režimu (je-li podporován architekturou procesoru). Synchronizační přerušení je následně použito pro aktivaci normálního běhu procesorů a jejích vzájemnou synchronizaci.



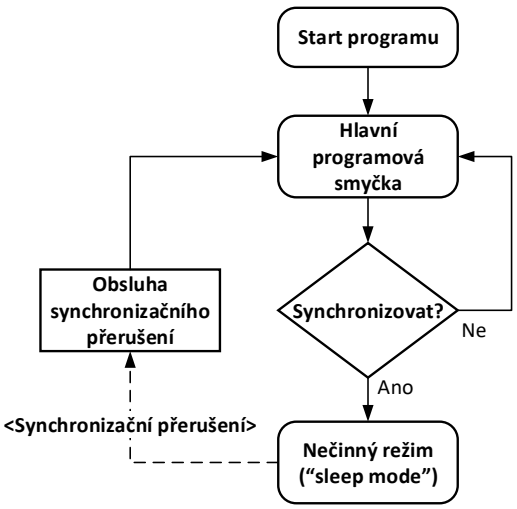
(a) Synchronizace pomocí restartu



(b) Synchronizace pomocí restartu s příznakem



(c) Synchronizace během obsluhy přerušení



(d) Synchronizace při přerušení klidového režimu

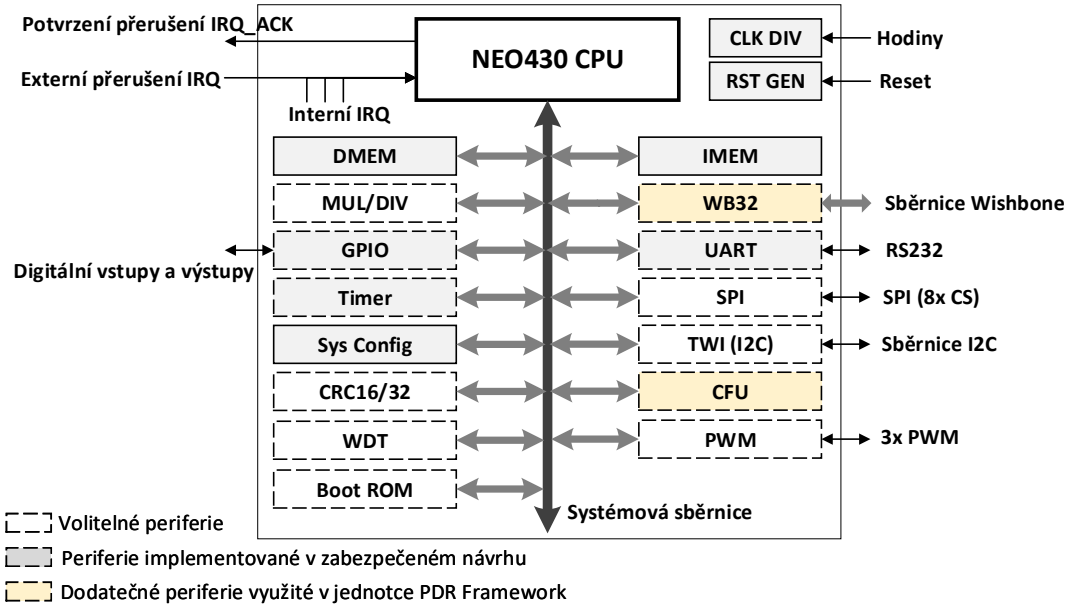
Obrázek 8.6: Způsoby programové implementace synchronizace procesoru

8.2 Návrh a implementace procesorového systému odolného proti poruchám

V této kapitole je popsán způsob zabezpečení mikrokontroleru NEO430 proti poruchám s využitím architektury ReSyTMR a samotný návrh vybraných metod synchronizace stavu procesoru, které byly poté implementovány a experimentálně ověřeny.

8.2.1 Mikrokontroler NEO430

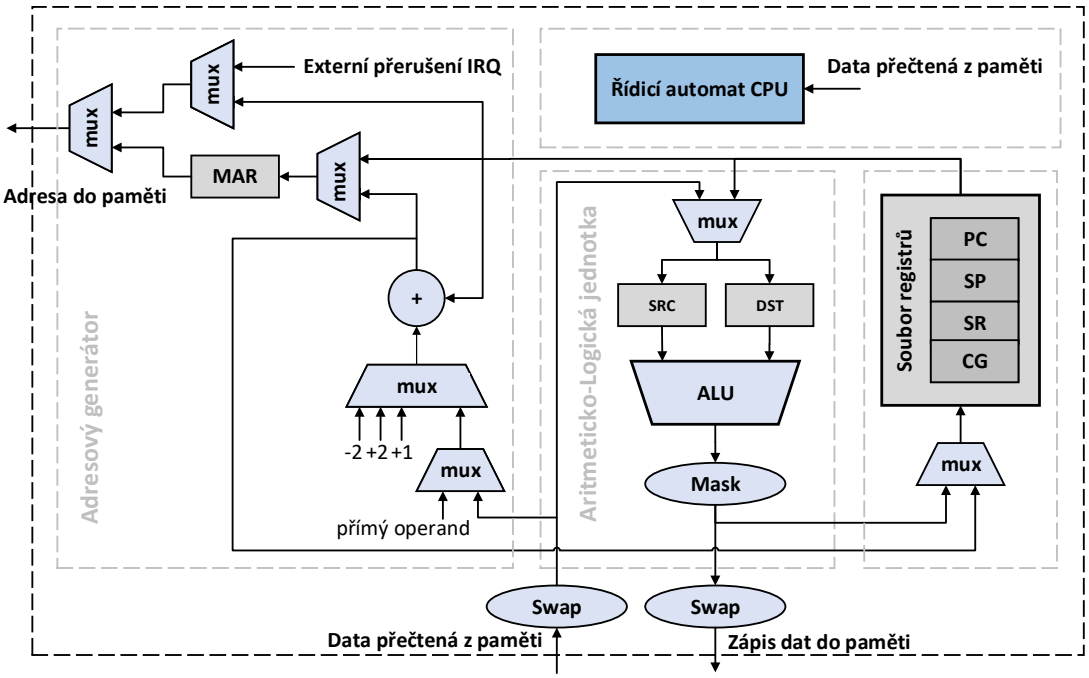
Mikrokontroler NEO430[1] je syntetizovatelná komponenta obsahující jádro 16-bitového procesoru NEO430, který je kompatibilní s procesorem TI MSP430 [72]. Procesor má harvardskou architekturu zahrnující oddělenou programovou (IMEM) a datovou (DMEM) operační paměť s konfigurovatelnou velikostí. Mikrokontroler NEO430 disponuje také různými základními periferními zařízeními, se kterými je jádro procesoru NEO430 propojeno přes systémovou sběrnici. Blokové schéma mikrokontroleru je znázorněno na obrázku 8.7.



Obrázek 8.7: Blokové schéma mikrokontroleru NEO430

Architektura jádra procesoru NEO430 je znázorněna na obrázku 8.8. Architektura procesoru se skládá z aritmeticko logické jednotky ALU (Arithmetic Logic Unit), řídicího arbitru procesoru CA (Control Arbiter), jednotky adresového generátoru AG (Address Generator) a souboru registrů. Provádění instrukcí v jádře procesoru NEO430 je založeno na principu zpracování několika mikrooperací, ze kterých se instrukce procesoru skládají. Pro dokončení provádění jedné instrukce je potřeba několika po sobě jdoucích cyklů. Provádění instrukcí je řízeno řídicím stavovým automatem, který generuje všechny řídicí signály zpracované v datové cestě při provádění mikrooperací. Řídicí automat procesoru po restartu přechází do stavu *IFETCH0*, ve kterém zpracování všech instrukcí začíná.

[1] V této disertační práci byla použita verze procesoru 0x0200. V době dokončování této práce je dostupná verze 0x0407. Projekt mikrokontroleru je dostupný na adrese `<https://github.com/stnolting/neo430>`.

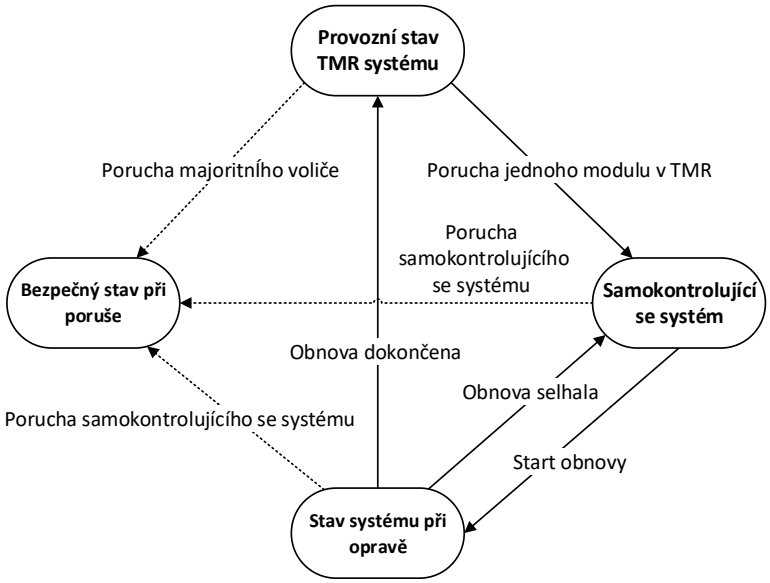


Obrázek 8.8: Architektura jádra procesoru NEO430

8.3 Návrh zabezpečené architektury ReSyTMR

Architektura *ReSyTMR (Reconfigurable Synchronizable TMR)* implementuje architekturu typu CG-TMR pro zabezpečení číslicového systému spolu s doplňujícími číslicovými obvody, které umožňují řízení procesu detekce poruchy a opravy stavu systému z poruchy pomocí rekonfigurace a synchronizace. Návrh zabezpečení s využitím architektury ReSyTMR uvažuje poruchové stavy systému a přechody mezi těmito stavy znázorněné stavovým diagramem na obrázku 8.9. Tento diagram znázorňuje přechod stavu systému z bezporuchového stavu do stavů, ve kterých se zabezpečený systém snaží provést opravu poruchy a dosáhnout zotavení do opět bezporuchového stavu. Bližší význam stavů architektury ReSyTMR je následující:

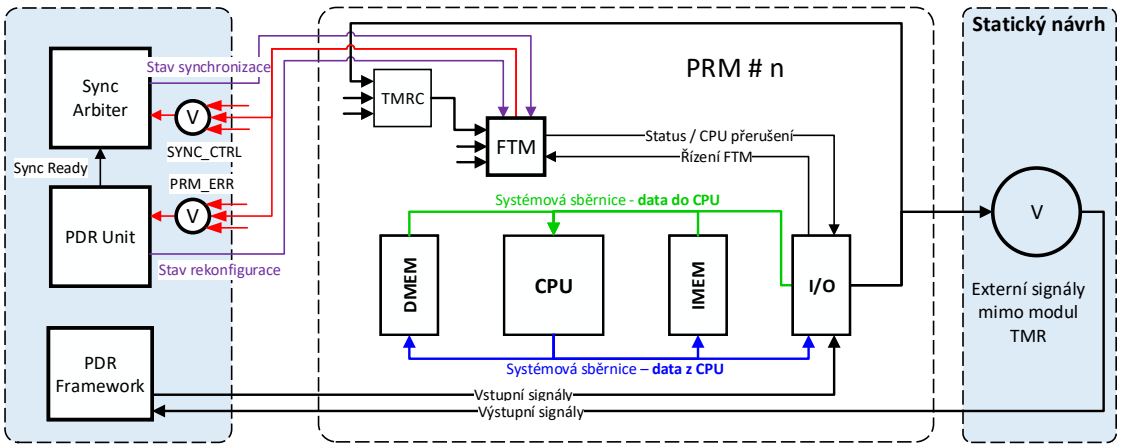
- *Bezporuchový stav TMR* – je výchozí provozní stav systému TMR, ve kterém jsou všechny výstupy majoritního voliče a kdy se dá předpokládat, že systém pracuje bez poruchy. Systém se navrací do tohoto stavu také v případě úspěšné opravy redundantního modulu, ve kterém byla detekována.
- *Samo-kontrolující se systém* – je degradovaný stav systému, ve kterém jsou bezporuchové pouze dva ze tří redundantních modulů. Systém TMR přechází do tohoto stavu v případě detekce poruchy v jednom z redundantních modulů.
- *Stav systému při opravě* – je degradovaný stav systému, ve kterém je provedena oprava redundantního modulu s detekovanou poruchou. Ostatní dva bezporuchové moduly jsou během opravy stavu třetího modulu stále v provozu.
- *Bezpečný stav při poruše* – je bezpečný stav, do kterého systém přejde v případě poruchy majoritního voliče vedoucí k selhání celého systému TMR nebo při výskytu poruchy v jednom ze dříve funkčních redundantních modulů.



Obrázek 8.9: Stavový diagram uvažovaného systému opravitelného z poruchy

Implementace prvků architektury ReSyTMR

Schéma jednoho modulu architektury ReSyTMR je znázorněno na obrázku 8.10. Na tomto schématu je ukázaná implementace propojení jednotlivých logických obvodů potřebných pro detekci poruch a řízení procesu opravy zabezpečeného procesoru NEO430 z poruchy. Každý modul TMR obsahuje stejné komponenty, které jsou vzájemně propojeny a tedy taktéž ztrojeny v rámci CG-TMR. Jeden modul architektury ReSyTMR obsahuje zabezpečený procesor NEO430, komparátor výstupních signálů TMRC a monitor stavu odolného systému FTM. Komparátor TMRC má zpětně přivedeny výstupy ze všech tří redundantních modulů. Informace o detekované poruše je předána jednotce FTM, která je propojena přes digitální vstupy a vstup externího přerušení s procesorem.



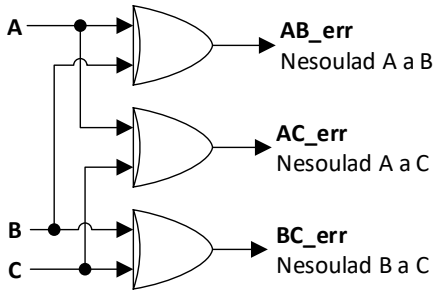
Obrázek 8.10: Základní schéma modulu odolné architektury ReSyTMR

Komparátor TMR

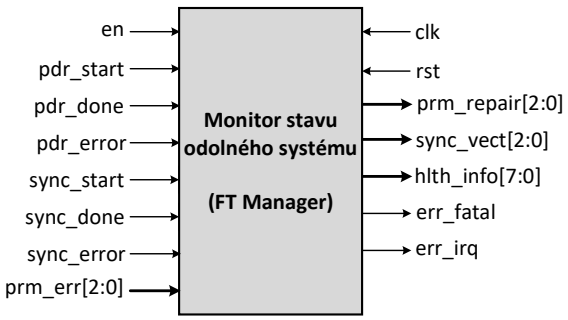
Komparátor *TMRC (TMR Comparator)* na rozdíl od majoritního voliče, který na základě majority vybírá správný výstup z TMR, provádí porovnání výstupů TMR a určuje, ve kterém modulu TMR se nachází porucha. Logika komparátoru implementovaná pomocí logických hradel XOR je znázorněna na obrázku 8.11. Výstupy komparátoru TMRC mohou být vyhodnoceny podle tabulky 8.2. Detekce poruchy minoritou nebo detekce poruchy všech modulů znamená selhání funkce zabezpečovací logiky. V takovém případě nemůže systém dále vykonávat svou funkci a musí přejít do bezpečného stavu. Jestliže je detekována porucha pouze v jednom z modulů architektury ReSyTMR, pak je systém schopný celkového zotavení s využitím opravných mechanismů rekonfigurace a synchronizace stavu.

Nesoulad23	Nesoulad13	Nesoulad12	Typ poruchy
0	0	0	Bez poruchy
0	0	1	Fatální porucha logiky
0	1	0	Fatální porucha logiky
0	1	1	Porucha modulu TMR 1
1	0	0	Fatální porucha logiky
1	0	1	Porucha modulu TMR 2
1	1	0	Porucha modulu TMR 3
1	1	1	Fatální porucha

Tabulka 8.2: Logická funkce komparátoru TMRC



Obrázek 8.11: Logika komparátoru TMR



Obrázek 8.12: Jednotka FT Manager

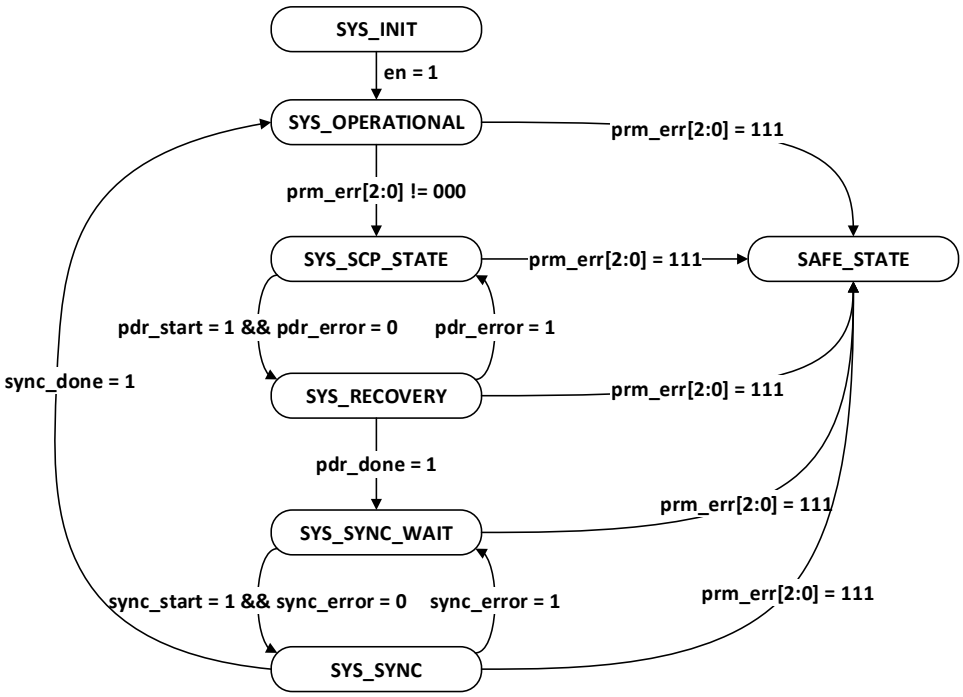
Monitor stavu odolného systému

Monitor stavu odolného systému *FTM (Fault Tolerant Monitor)* byl navržen pro řízení stavu systému odolného proti poruchám v rámci architektury ReSyTMR. Blokové schéma jednotky FTM je znázorněno na obrázku 8.12. Funkce jednotky FTM byla implementována ve smyslu stavového diagramu znázorněného na obrázku 8.9.

Důvodem pro implementaci jednotky FTM v architektuře ReSyTMR je umožnit řízení reakce zabezpečeného systému na poruchové podmínky, které mohou za jeho běhu vzniknout. Vstup *prm_err* jednotky FTM je propojen s výstupem jednotky TMRC, která porovnává výstupy všech redundantních modulů a případnou poruchu indikuje chybovým

vektorem. Současně je FTM schopný zareagovat na selhání procesu rekonfigurace, indikované na vstup *pdr_error* řadičem rekonfigurace, a selhání procesoru synchronizace stavu indikované na vstup *sync_error* arbitrem synchronizace.

Detailnější pohled na implementovaný stavový automat je poskytnut obrázkem 8.13. Bezporuchový provozní stav systému je SYS_OPERATIONAL. Systém v případě detekce poruchy jednoho modulu přechází do stavu SYS_SCP_STATE. Jednotka FTM v tomto stavu aktivuje signál *prm_repair*, který je připojen k řadiči rekonfigurace GPDRC a indikuje požadavek na start rekonfigurace vybraného modulu PRM. Po startu rekonfigurace, který je indikován řadičem rekonfigurace na vstup *pdr_start* jednotky FTM, přechází FTM do stavu SYS_RECOVERY. FTM v tomto stavu čeká na dokončení rekonfigurace, které bude indikováno na vstup *pdr_done*. Následně FTM přechází do stavu SYS_SYNC_WAIT, kde čeká na start synchronizace stavu modulů v CG-TMR. Provedení procesu synchronizace stavu v systému je řízeno arbitrem synchronizace, který na vstup *sync_start* a *sync_done* jednotky FTM indikuje začátek a konec synchronizace. V případě poruchy, detekované během procesů rekonfigurace nebo synchronizace, se jednotka FTM vrací do předchozího stavu pro opakování procesu.

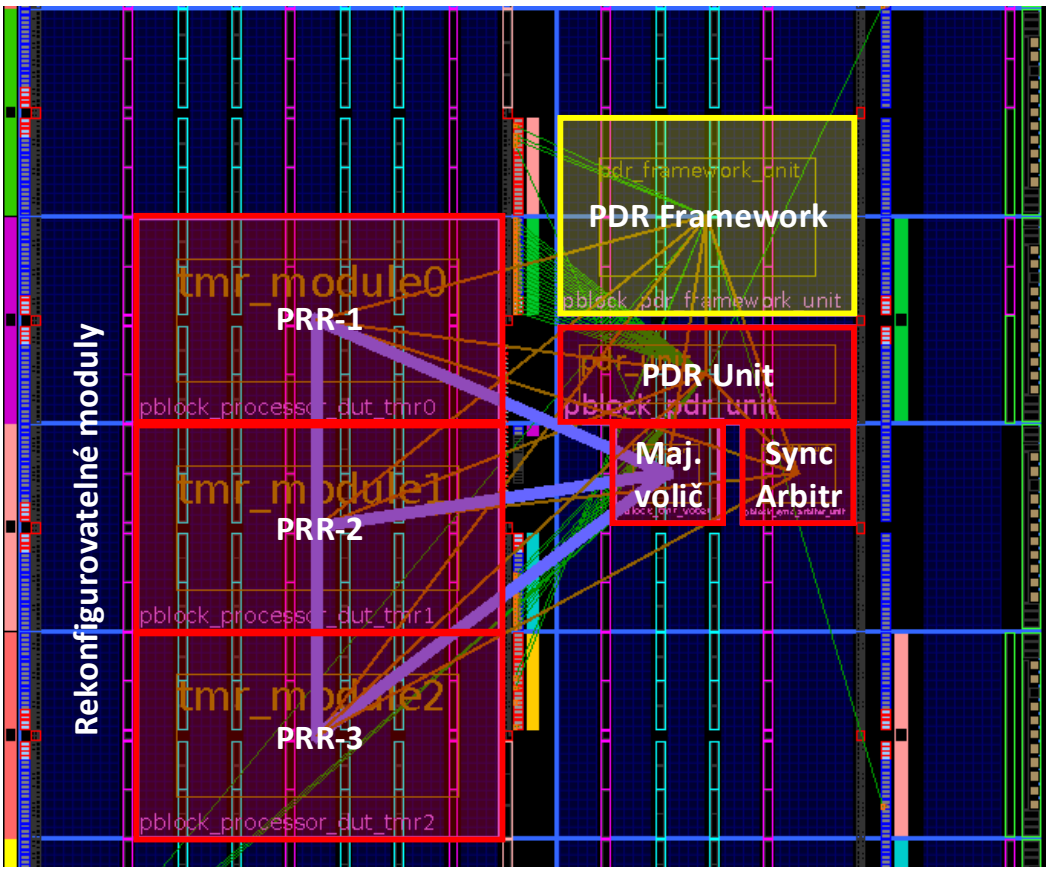


Obrázek 8.13: Implementace stavového automatu jednotky FTM

Jednotka FTM může řídit celý proces opravy a obnovy stavu automaticky nebo komunikovat se zabezpečeným systémem (tedy procesorem), který provede rozhodnutí na základě vytížení aktuálně prováděných úloh a stavu okolního prostředí. Při verifikaci správné funkce systému odolného proti poruchám bylo pro řízení jednotlivých kroků procesu obnovy stavu systému použito testovací prostředí PDR Framework, popsané v kapitole 8.5.1. Díky jednotce FTM lze monitorovat stav odolného systému a poskytnout bližší informace o chování systému v přítomnosti poruch.

8.3.1 Rekonfigurovatelná architektura

Rozvržení rekonfigurovatelné architektury ReSyTMR v konfigurační paměti obvodu FPGA je znázorněno na obrázku 8.14. Zobrazený rekonfigurovatelný návrh implementuje základní architekturu ReSyTMR navrženou podle schématu z obrázku 8.10. V této verzi návrhu obsahuje každý rekonfigurovatelný modul ReSyTMR všechny komponenty mikrokontroleru NEO430 (na rozdíl od návrhu se sdílenou datovou pamětí). Statická část návrhu implementuje jednotku PDR Unit, majoritní volič a arbitra synchronizace. Tyto komponenty jsou součásti návrhu systému odolného proti poruchám. Dále je ve statické části návrhu umístěna jednotka PDR Framework, která slouží pro komunikaci se zabezpečeným systémem během experimentů a pro monitorování procesů rekonfigurace a synchronizace stavu. Rekonfigurovatelná oblast je rozdělena do tří rekonfigurovatelných regionů PRR-1, PRR-2 a PRR-3. Každý rekonfigurovatelný region obsahuje jednu instanci modulu architektury ReSyTMR, jejíž výstupy jsou přivedeny do redukujícího majoritního voliče. Rozhraní mezi rekonfigurovatelnými moduly a obvody ve statické části návrhu je tvořena proxy logikou, která je automaticky implementována návrhovými nástroji od firmy Xilinx.



Obrázek 8.14: Rozvržení rekonfigurovatelné architektury ReSyTMR v FPGA

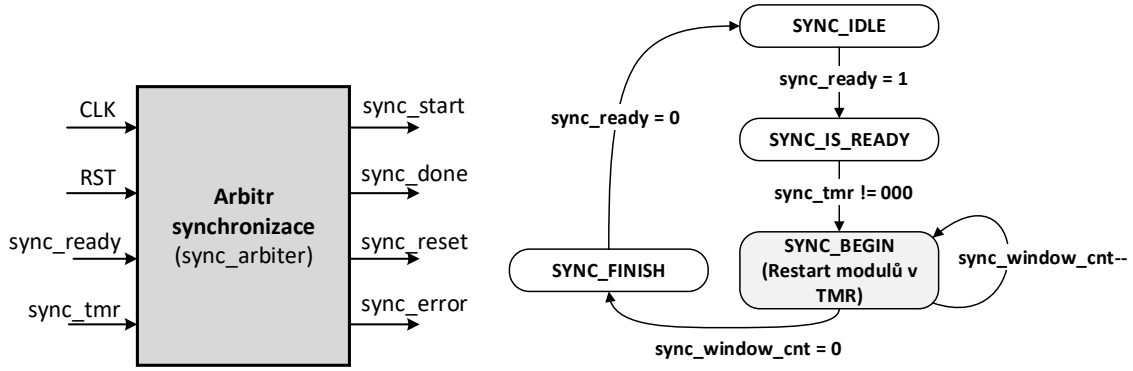
8.4 Implementace metod pro synchronizaci

Na základě myšlenek představených v kapitole 8.1 a vyhodnocení vhodnosti možných řešení synchronizace stavu pro mikrokontroler s jádrem procesoru NEO430, zabezpečený pomocí architektury ReSyTMR, byly navrženy a implementovány dvě metody synchronizace stavu procesoru využívající synchronizační restart a sdílenou paměť.

8.4.1 Synchronizace pomocí synchronizačního restartu

Pro implementaci metody synchronizaci pomocí synchronizačního restartu nebylo nutné provádět žádné změny v základní architektuře odolné architektury ReSyTMR prezentované na obrázku 8.10. Blokové schéma implementované jednotky arbitra synchronizace je znázorněno na obrázku 8.15a.

Jednotka arbitru pracuje podle stavového automatu uvedeného na obrázku 8.15b. Řadič rekonfigurace GPDRC jednotky PDR Unit po dokončení rekonfigurace indikuje připravenost na synchronizaci pomocí svého výstupu *pdr_sync*. Tento výstup je přiveden na vstup *sync_ready* arbitra synchronizace. Vstupní vektor *sync_tmr* pro spuštění synchronizace je přiveden z jednotky FTM ze všech redundantních modulů. V jednotce arbitra synchronizace je možné nastavit velikost časového okna *sync_window_cnt* udávající, jak dlouho bude signál restartu modulů *sync_reset* aktivní. Po deaktivaci tohoto signálu začínají restartované redundantní moduly opět pracovat synchronně.



(a) Blokové schéma

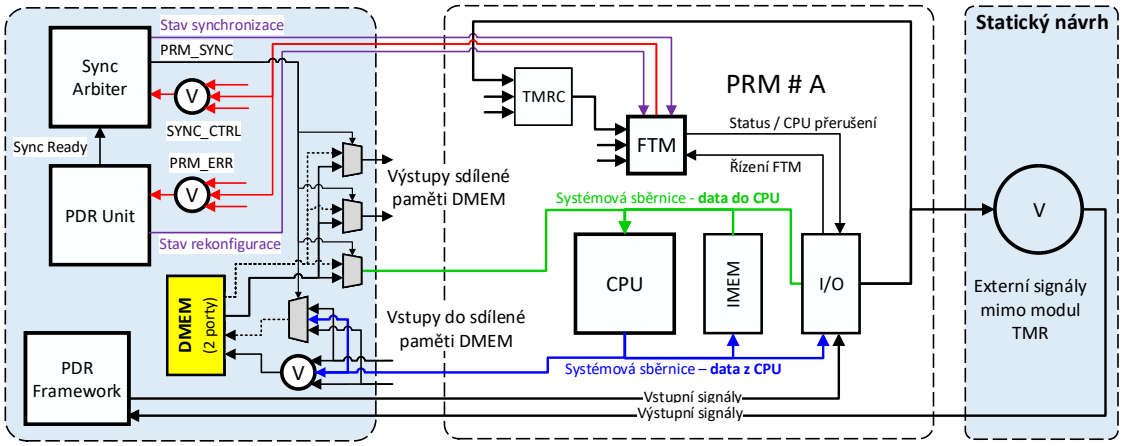
(b) Stavový automat

Obrázek 8.15: Implementace jednotky arbitru synchronizace pro synchronizační restart

Každý modul architektury ReSyTMR je schopný nezávisle detekovat poruchu jednoho z modulů pomocí jednotky TMRC. Porucha je indikována jednotce FTM, která umožní na základě aktuálního stavu specifickou opravnou reakci. Zabezpečený procesor může na základě aktuálního vytížení rozhodnout o provedení rekonfigurace. Rekonfigurace je provedena jednotkou PDR Unit. Dokončení rekonfigurace modulu PRM, ve kterém byla detekována porucha, je indikováno arbitru synchronizace. Arbitr synchronizace řídí proces synchronizace. Arbitr provede synchronizaci pomocí synchronního restartu logiky všech redundantních modulů.

8.4.2 Synchronizace pomocí sdílené datové paměti

Schéma architektury ReSyTMR s upraveným návrhem mikrokontroleru se sdílenou datovou pamětí je znázorněno na obrázku 8.16. Datová paměť procesoru byla implementována ve statické části číslicového návrhu. V každém redundantním modulu má procesor k dispozici instrukční paměť IMEM a periferní zařízení v modulu I/0. Tyto komponenty jsou vzájemně propojeny systémovou sběrnicí mezi sebou a také s datovou pamětí DMEM, která je sdílená.



Obrázek 8.16: Schéma modulu odolné architektury ReSyTMR se sdílenou datovou pamětí

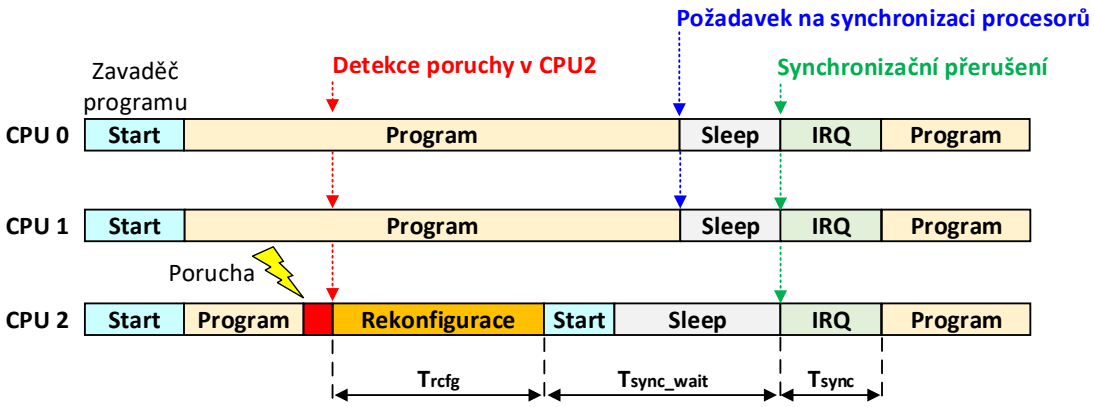
Vyřešení konfliktů v přístupu do sdílené paměti

Při implementaci sdílené paměti procesoru bylo nutné vyřešit konflikt mezi přístupy do paměti z bezporuchových procesorů a přístupy do paměti z opravovaného procesoru, který má být synchronizován. Další problém se týká vystavení hodnot na systémovou sběrnici v momentě, kdy majorita procesorů komunikuje např. s datovou pamětí a synchronizovaný procesor potřebuje komunikovat se svými periferními zařízeními. V takové případě byly na sběrnici vystaveny jak adresovaná data z datové paměti, tak hodnoty registrů z I/O modulu. Popsané konflikty byly vyřešeny implementací druhého přístupového portu ke sdílené paměti DMEM, který umožňuje synchronizovanému procesoru pouze čtení dat. Přístup do sdílené paměti je tedy řízen multiplexory, které umožňují bezporuchovým procesorům normální přístup do paměti a oddělují tuto komunikaci od samotné systémové sběrnice synchronizovaného procesoru. Normální přístup do paměti je volen majoritně. Paměť na základě správných řídicích signálů vystaví adresovaná data zpět na sběrnici pro zpracování jádrem procesoru.

Synchronizace běhu procesorů pomocí synchronizačního přerušení

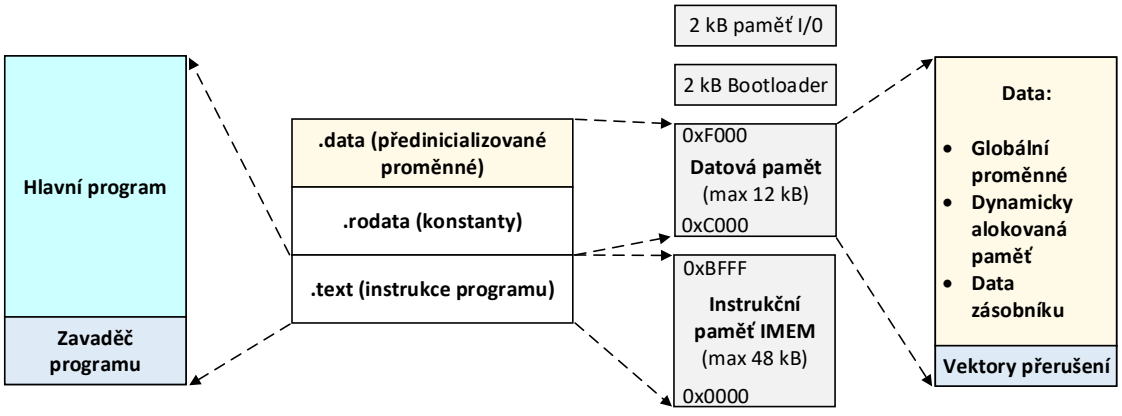
Způsob implementace synchronizace běhu procesorů pomocí synchronizačního přerušení je znázorněn na diagramu na obrázku 8.17. Protože je synchronizační přerušení vzhledem k provádění instrukcí procesorem vyvoláno asynchronní událostí, je nutné uvést redundantní procesory do klidového režimu (tzv. spící režim). Synchronizační přerušení pak může být použito pro probuzení procesorů, které bude provedeno již synchronně vzhledem k jejich předchozímu stavu. Přechod do klidového režimu umožní bezporuchovým procesorům do-

končit aktuálně rozpracované instrukce (resp. aktuální úlohu). V opačném případě by bez použití přechodu do klidového režimu mohlo dojít k situaci, že požadavek na synchronizační přerušení přijde ve chvíli, kdy se synchronizovaný procesor i bezporuchové procesory nachází v jiné fázi dokončení aktuálně prováděné instrukce.



Obrázek 8.17: Schéma synchronizace redundantních procesorů pomocí přerušení

Procesor po svém startu začíná provádět program zavaděče, který provádí nízkoúrovňové nastavení procesoru. Program zavaděče je implementovaný v jazyce Assembler a jeho kód je umístěn na adrese 0x0000 v instrukční paměti, odkud procesor začíná provádět program. Zavaděč inicializuje všechny procesorové registry, inicializuje programové prostředí procesoru a na konci zavolá hlavní funkci *main* programu. Při inicializaci programového prostředí je smazána celá datová paměť a všechny hodnoty sloužící pro nastavení předem inicializovaných globálních proměnných jsou zkopírovány z instrukční paměti IMEM do datové paměti DMEM. Schéma rozložení instrukcí a dat zkompilovaného programu s návazností na umístění v pamětech procesoru je znázorněno na obrázku 8.18.



Obrázek 8.18: Schéma rozložení programu v pamětech mikrokontroleru NEO430

Implementace programové synchronizace stavu

Za účelem provedení synchronizace rekonfigurovaného procesoru byl v rámci zavaděče implementován podprogram, který zajistí přechod rekonfigurovaného procesoru do klidového režimu, kde bude vyčkávat na synchronizační přerušení. Implementovaná část programu zavaděče je uvedena ve výpisu 8.1. Pro přechod do klidového režimu je nutná aktivace digitálního vstupu, na který je přiveden signál z arbitru synchronizace.

```
    mov &0xFFAC,r10       ; Vyčtení hodnoty digitálních vstupů
    and #0x8000,r10       ; Vymaskování hodnoty pro vstup 15
    tst r10               ; Test na 0
    jeq __no_sync         ; Přeskoč synchronizaci
    mov  #0, r2           ; Vymazání příznaků z~řídicího registru
    mov  #0x4700, &0xFFB8 ; Deaktivace jednotky watchdog
    mov  #(1<<3), r2      ; Povolení přerušení procesoru
    bis  #(1<<4), r2      ; Aktivace klidového režimu
    nop
__no_sync:                ; Pokračování kódu zavaděče
```

Výpis 8.1: Část programu zavaděče pro synchronizaci opravovaného procesoru

Před provedením samotné synchronizace stavu celého redundantního systému musí být bezporuchové procesory přivedeny taktéž do klidového režimu. To lze docílit aktivaci dalšího digitálního vstupu, jenž je v programu procesorů detekován. Synchronizační přerušení je vyvoláno aktivací signálu externího přerušení procesoru. Procesor se v klidovém režimu nachází ve stavu IFETCH0, ve kterém vyčkává na příchod přerušení. V momentě příchodu přerušení si každý procesor uloží do programového zásobníku návratovou adresu a hodnotu stavového registru. Následně je smazán příznak pro nastavení klidového režimu a zakázána další přerušení. Procesor na základě typu přerušení vyvolá odpovídající obslužnou rutinu z tabulky vektorů přerušení. Programový zásobník i tabulka vektorů přerušení se nachází ve sdílené datové paměti. Ve vyvolané obslužné rutině přerušení musí být provedena synchronizace všech synchronizovaných objektů procesoru. Po dokončení zpracování přerušení je obnoven stav stavového registru a programového čítače ze zásobníku. Pro synchronizovaný procesor to znamená, že začne provádět program tam, kde ostatní procesory skončily. Ukázka implementace obslužné rutiny synchronizačního přerušení je uvedena ve výpisu kódu 8.2.

```
void __attribute__((__interrupt__)) sync_irq_handler(void)
{
    /* Uložení hodnot registrů do sdílené datové paměti. */
    asm volatile ("mov r1,  &0xCF82");
    asm volatile ("mov r4,  &0xCF88");
    ...
    /* Obnovení hodnot registrů ze sdílené datové paměti. */
    asm volatile ("mov &0xCF82, r1");  // R1 - SP
    asm volatile ("mov &0xCF88, r4");  // R4
    ...
}
```

Výpis 8.2: Ukázka implementace obslužné rutiny synchronizačního přerušení

Ukázka ze simulace implementované metody synchronizace stavu registrů procesoru v obslužné rutině synchronizačního přerušení v programu Xilinx iSim je znázorněna na obrázku 8.19.

Obrázek 8.19: Simulace synchronizace registrů procesoru během obsluhy přerušení

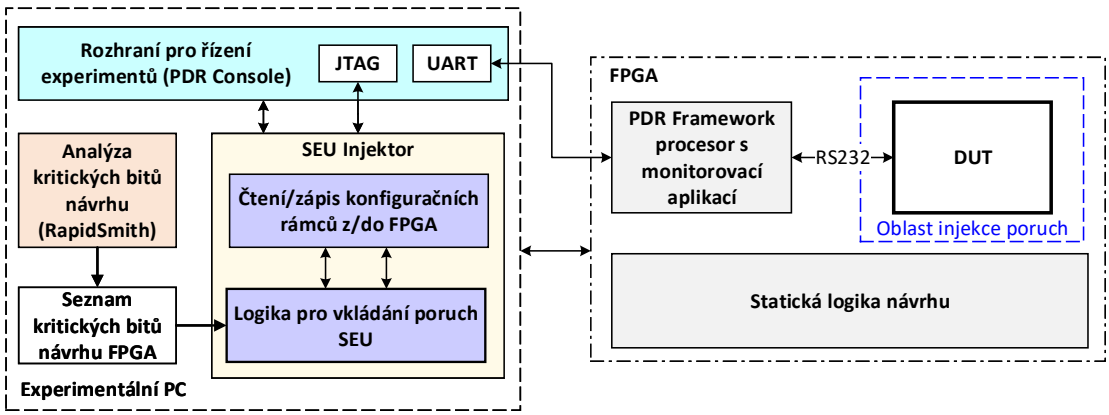
8.5 Experimentální a implementační výsledky

Tato kapitola popisuje způsob experimentálního ověření správné funkce implementovaných variant zabezpečeného procesoru NEO430 a všechny provedené experimenty, které byly zaměřeny na ověření spolehlivostních parametrů zabezpečeného systému při opravě a zotavení jeho stavu z poruchy za pomocí rekonfigurace a navržených metod synchronizace stavu.

8.5.1 Testovací prostředí

Experimentální a testovací prostředí, které bylo navržené pro experimenty se systémem řízeným mikrokontrolerem NEO430 a ověření spolehlivostních parametrů zabezpečených variant tohoto systému je znázorněno na obrázku 8.20. Toto prostředí se skládá z následujících komponent:

- *DUT (Design Under Test)* – DUT označuje cílový systém, pro něhož bylo implementováno několik variant zabezpečení a které jsou v této kapitole porovnány. Pro všechny varianty implementace systému je v konfigurační paměti vyhrazena stejná oblast, do které jsou během experimentů injektovány poruchy typu SEU.
- *PDR Framework* – Jednotka řízena procesorem NEO430 s testovacím programem, který umožňuje řízení a monitorování experimentů. Procesor v jednotce je propojen přes sériové rozhraní s ověřovaným návrhem DUT. Procesory mezi sebou mohou vzájemně komunikovat. PDR Framework umožňuje jak komunikaci s PC pomocí sériového rozhraní, tak komunikaci s cílovým návrhem zabezpečeného systému.
- *PDR Console* – Programový nástroj, který umožňuje komunikovat z PC s jednotkou PDR Frameworku za pomocí komunikačního sériového protokolu.
- *SEU Injektor* – Programový nástroj, který umožňuje automaticky vkládat poruchy typu SEU z PC za pomocí rozhraní JTAG. Tento nástroj byl popsán v kapitole 5.4.
- *RapidSmith* – Programový nástroj použitý pro analýzu cílového číslicového návrhu pro FPGA a zjištění polohy kritických bitů návrhu v konfigurační paměti FPGA.



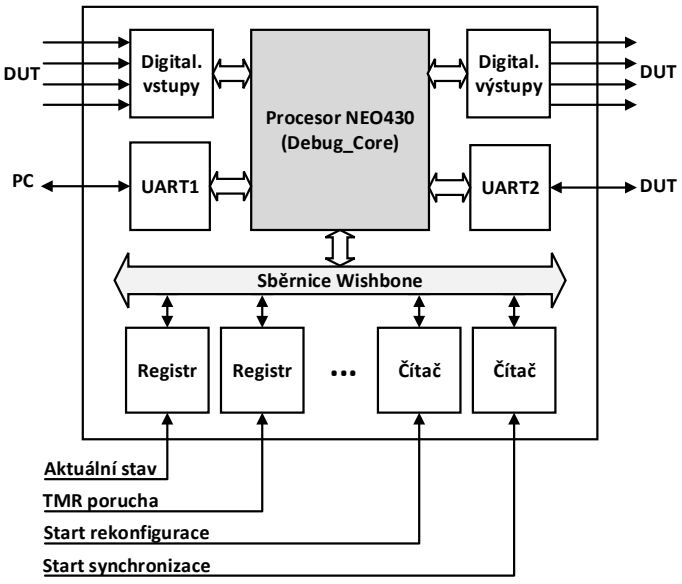
Obrázek 8.20: Schéma experimentálního prostředí pro ověření zabezpečení procesoru

Experimentální prostředí PDR Framework

Pro možnost verifikace funkce systému odolného proti poruchám, ověření správné funkce provedení rekonfigurace poruchového modulu v architektuře ReSyTMR a jeho synchronizace s ostatními redundantními moduly bylo navrženo a implementováno experimentální prostředí *PDR Framework*. Pro návrh tohoto experimentálního prostředí byly stanoveny následující základní požadavky:

- Podpora jednoduchého protokolu pro komunikaci mezi procesory a PC.
- Podpora uživatelského prostředí za pomocí příkazového řádku a komunikace po RS232.
- Podpora operací pro řízení a verifikaci detekce poruch, rekonfigurace a synchronizace.
- Řízení specifických operací z PC jak v jednotce PDR Framework, tak v DUT.
- Implementace registrů pro záchyt různých typů signálů z DUT a logiky ReSyTMR.

Schéma jednotky PDR Framework je znázorněno na obrázku 8.21. PDR Framework je taktéž řízen mikrokontrolerem s jádrem procesoru NEO430. Proto, aby bylo možné řídit experimenty pomocí uživatelského terminálu nebo pomocí navrženého komunikačního protokolu a zároveň aby bylo možné provádět komunikaci s procesorem v DUT, má mikrokontroler NEO430 implementován dvě sériová rozhraní. Mikrokontroler standardně podporuje pouze jedno sériové rozhraní. Návrh procesoru musel být tedy rozšířen. Dále PDR Framework implementuje sadu datových registrů a čítačů, které umožňují zachytit vstupní signály připojené z verifikovaného číslicového návrhu. Tyto registry a čítače jsou přístupné pomocí sběrnice Wishbone. Digitální vstupy a výstupy jednotky PDR Framework mohou být použity pro komunikaci s DUT nebo pro řízení logiky experimentů v rámci architektury ReSyTMR.



Obrázek 8.21: Schéma architektury jednotky PDR Framework

Pro komunikaci mezi procesorem NEO430 v jednotce PDR Framework a procesorem NEO430 v DUT byl implementován jednoduchý sériový komunikační protokol, který umožňuje experimentální aplikaci v jednotce PDR Framework řídit funkci procesoru v DUT nebo vyčítat jeho vnitřní stavové registry.

Experimentální aplikace implementovaná pro procesor NEO430 v jednotce PDR Framework podporuje dva různé komunikační režimy. Aplikace implementuje komunikační protokol pro podporu komunikace s procesorem v DUT. Tento způsob komunikace je využit také při provádění automaticky řízených experimentů. Díky tomuto protokolu je testovací aplikace spuštěná na experimentálním PC schopna komunikovat s jednotkou PDR Framework a také zprostředkovaně s procesorem v DUT. Alternativně může být aplikace zkompilována s podporou uživatelského terminálu, ve kterém lze pomocí jednoduchých příkazů řídit a sledovat stav experimentů. Přehled vybraných příkazů uživatelského terminálu jednotky PDR Framework je uveden v tabulce 8.3.

Příkaz	Popis funkce příkazu
pdr status	Zobrazí aktuální stav a výsledky procesu rekonfigurace.
pdr reconf <PRM>	Spustí rekonfiguraci vybrané oblasti PRM.
sync status	Zobrazí aktuální stav a výsledky procesu synchronizace stavu.
sync reset	Provede synchronizační restart.
tmr status	Zobrazí aktuální stav TMR.
dut status	Zobrazí aktuální stav DUT.
dut errst	Zobrazí detailní přehled o poruchách v DUT.

Tabulka 8.3: Přehled vybraných příkazů uživatelského terminálu jednotky PDR Framework

8.5.2 Nároky na implementaci architektury ReSyTMR

Implementační režie jednoho modulu architektury ReSyTMR a jeho komponent je uvedena v tabulce 8.5. Schéma modulu a jeho propojení se statickou logikou v rámci rekonfigurovatelného návrhu je znázorněno na obrázku 8.10 v kapitole 8.2.

Virtex5 XC5VSX50T Implementované komponenty	Počet registrů	Počet LUT	Počet BMEM	Počet DMEM
Jádro procesoru (neo430_cpu)	164	439	0	16
Datová paměť (neo430_dmem)	1	18	2	0
Digitální V/V (neo430_gpio)	81	30	0	16
Instrukční paměť (neo430_imem)	1	20	4	0
Jednotka násobení/dělení(neo430_muldiv)	116	122	0	1
Systémová konfigurace (neo430_sys_config)	14	12	0	0
Časovač (neo430_timer)	55	67	0	0
Sériový port UART (neo430_uart)	88	89	0	1
Řadič sběrnice Wishbone (neo430_wb32)	118	64	0	0
Ostatní logika	27	59	0	1
Mikrokontroler (celkem)	665	920	6	35

Tabulka 8.4: Velikost implementace mikrokontroleru s procesorem NEO430 v FPGA

Virtex5 XC5VSX50T Implementované komponenty	Počet registrů	Počet LUT	Počet BMEM	Počet DMEM
Mikrokontroler NEO430	665	921	6	35
Jednotka FTM	6	21	–	–
Jednotka TMRC	–	93	–	–
Modul ReSyTMR (celkem)	671	1036	6	35

Tabulka 8.5: Velikost modulu ReSyTMR a jeho komponent v FPGA

Porovnání implementačních nároků nezabezpečeného procesoru NEO430 a zabezpečených variant implementace procesoru pomocí architektury FG-TMR a CG-TMR je uvedeno v tabulce 8.6. V tabulce je vidět, že použití FG-TMR vede ke značnému snížení pracovní frekvence systému. Důvodem je implementace zpětných vazeb na úrovni každého registru, což vede k prodloužení kritických cest v návrhu systému. Současně, také implementace architektury ReSyTMR vede k podobnému snížení pracovní frekvence. Zde je důvodem připojení všech výstupů redundantních modulů na vstupy komparátoru TMR, který je umístěn v každém modulu.

Varianta systému	FF	LUT	Max. frekvence [MHz]
Nezabezpečený procesor	1831	2876	95.229
Zabezpečení pomocí FG-TMR	4231	10205	51.674
Zabezpečení pomocí CG-TMR	2010	3243	87.1
ReSyTMR s sync. restartem	2013	3308	80.159

Tabulka 8.6: Porovnání režie implementace systému při různých způsobech zabezpečení

8.5.3 Vyhodnocení odolnosti proti poruchám

Pro ověření odolnosti proti poruchám navrženého řešení systému mikrokontroléru řízeného procesorem NEO430 a za účelem porovnání dosažených spolehlivostních parametrů s alternativními metodami zabezpečení systému byly provedeny experimenty, během kterých byla provedena injekce poruch (podle postupu popsaného v kapitole 5.4) do konfigurační paměti FPGA s následujícími variantami různě zabezpečeného systému:

a) *Simplex* – nezabezpečený systém mikrokontroléru,

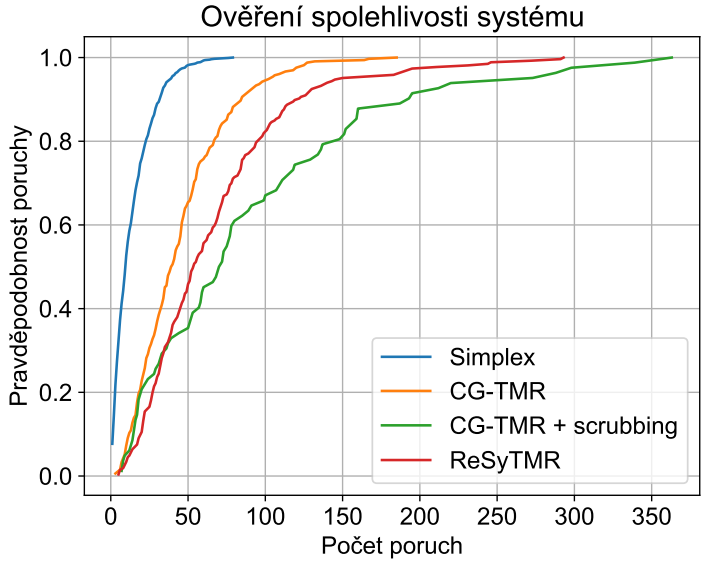
b) *CG-TMR* – systém zabezpečený pomocí CG-TMR (implementace mikrokontroléru byla v číslicovém návrhu ztrojena, majoritní volič byl implementován mimo oblast injektovaných poruch v konfigurační paměti FPGA),

c) *FG-TMR* – systém zabezpečený pomocí FG-TMR (všechny registry mikrokontroléru byly pomocí nástroje BL-TMR automaticky zabezpečeny pomocí architektury DTMR se synchronizujícím voličem),

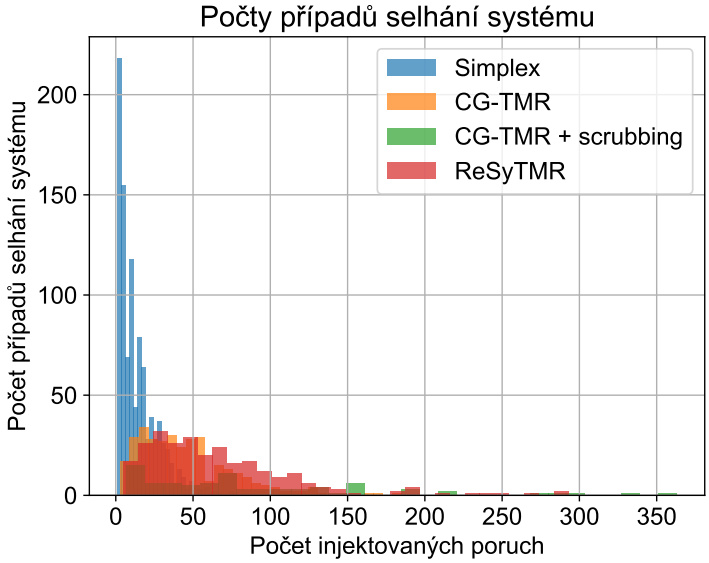
d) *CG-TMR + scrubbing* – systém zabezpečený pomocí CG-TMR s periodickým čištěním konfigurační paměti. Periodické čištění bylo simulováno pomocí programu v PC, který po každém cyklu 20 injektovaných poruch přeprogramoval oblast s návrhem CG-TMR původním konfigurací.

e) *ReSyTMR* – systém zabezpečený pomocí architektury ReSyTMR se synchronizací stavu pomocí synchronizačního restartu.

Porovnání spolehlivosti implementace nezabezpečeného návrhu mikrokontroleru s procesorem NEO430, návrhu zabezpečeného pomocí architektury CG-TMR, varianty CG-TMR se simulovaným periodickým čištěním konfigurační paměti a návrhu zabezpečeného pomocí architektury ReSyTMR je znázorněno na obrázku 8.22.



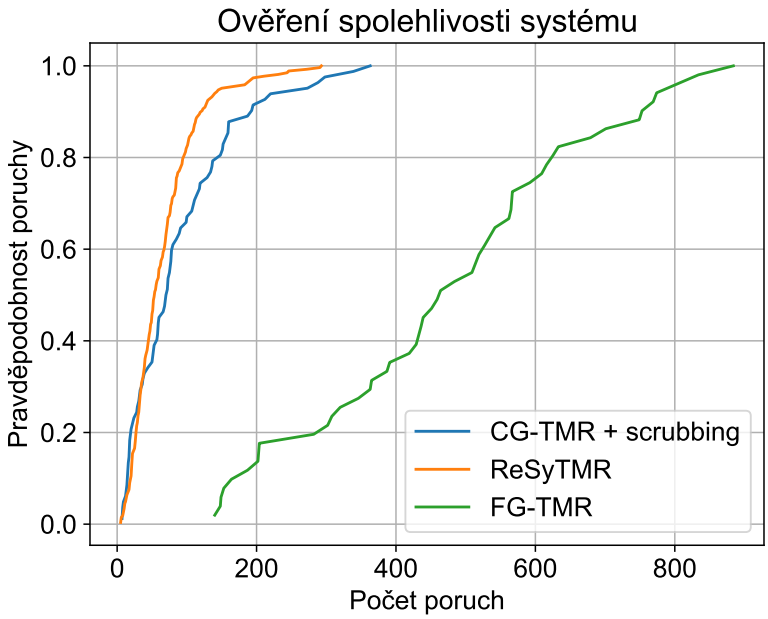
(a)



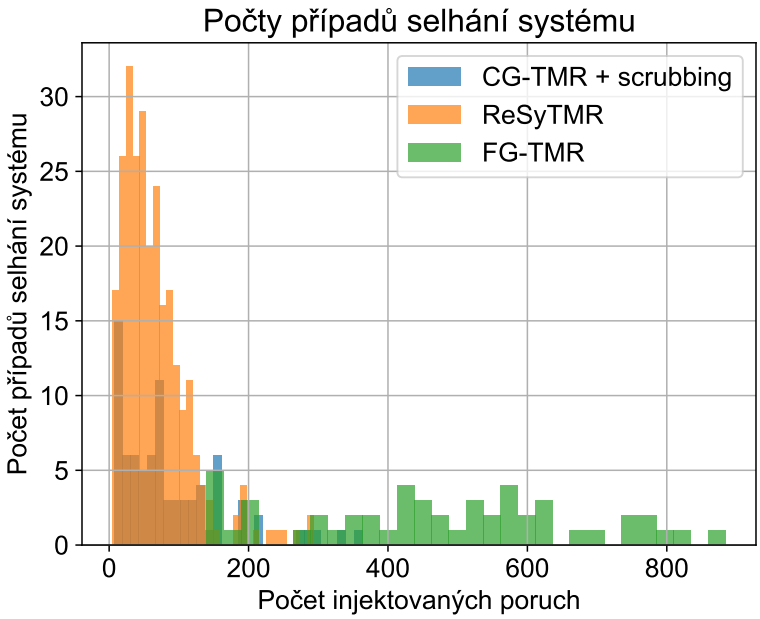
(b)

Obrázek 8.22: Porovnání spolehlivosti nezabezpečeného systému, CG-TMR, CG-TMR s periodickým čištěním a ReSyTMR

Porovnání spolehlivosti implementace návrhu zabezpečeného pomocí architektury ReSyTMR a návrhu zabezpečeného pomocí FG-TMR je znázorněno na obrázku 8.23.



(a)



(b)

Obrázek 8.23: Porovnání spolehlivosti CG-TMR s periodickým čištěním, ReSyTMR a FG-TMR

Spolehlivost dosažená implementací FG-TMR pro každý registr sekvenční logiky číslicového návrhu je daleko větší než spolehlivost návrhu, který využívá hrubozrnnou architekturu CG-TMR. Z výsledků je patrné zlepšení spolehlivosti díky implementaci opravného

mechanismu. Ovšem velkou nevýhodou implementace FG-TMR je vliv na maximální pracovní frekvenci číslicového návrhu. Podle tabulky 8.6 dochází až k 50% poklesu maximální pracovní frekvence.

K výsledkům experimentů s injekcí poruch typu SEU je nutné dodat, že díky analýze kritických bitů číslicového návrhu byly poruchy vkládány pouze do užitečných bitů konfigurační paměti FPGA. Tento způsob injekce poruch neslouží k simulaci reálných podmínek při působení kosmického záření nýbrž hlavně k ověření opravných mechanismů implementovaných v ověřovaném systému odolném proti poruchám.

8.6 Shrnutí

Při hodnocení možností pro návrh synchronizace stavu různých procesorů je důležité analyzovat, jak lze provést synchronizaci pro jednotlivé synchronizované objekty definující stav procesoru. Stav procesoru byl popsán obecně v kapitole 8.1.1. Jak již bylo řečeno, architektury procesorů se mohou lišit svou komplexností a odlišnostmi v implementaci některých funkčních komponent. Možností pro návrh synchronizace stavu pro v této práci zmíněné procesory Xilinx PicoBlaze a Xilinx MicroBlaze jsou porovnány s mikrokontrolérem NEO430 v tabulce 8.7.

Synchronizované objekty	NEO430	PicoBlaze	MicroBlaze
Programový čítač	R/W	Není přístupné	R/W
Ukazatel na zásobník	R/W	Není přístupné	R/W
Programový zásobník	R/W	Není přístupné	R/W
Všeobecné registry	R/W	R/W	R/W
Řídicí registry	R/W	R/W	R/W
Stavové registry	R/W	Není přístupné	R/W
Datová paměť	R/W	R/W	R/W

Tabulka 8.7: Porovnání přístupu k synchronizovaných objektům u různých procesorů (R – čtení, W – zápis)

Kapitola 9

Závěr

Závěrečná kapitola této disertační práce shrnuje výsledky prezentovaných výzkumných aktivit a popisuje hlavní přínos práce. V kapitole jsou také shrnuty výzkumné otázky, které celou disertační práci provázely. Dále jsou v této kapitole popsány možná rozšíření práce a směry, kterými by se výzkum v oblasti návrhu rekonfigurovatelných systémů využívajících synchronizaci stavu pro opravu stavu rekonfigurovaného redundantního modulu v architektuře TMR mohl dále ubírat.

9.1 Shrnutí výsledků práce

V rámci této disertační práce byla představena metodika pro návrh a implementaci číslicových obvodů pro synchronizaci stavu rekonfigurovatelných modulů TMR architektury systému odolného proti poruchám implementovaného do FPGA. Synchronizace stavu je jednou z možností pro provedení opravy stavu číslicového systému zabezpečeného pomocí architektury TMR po vzniku poruchy. V této práci je uvažováno použití techniky synchronizace stavu spolu s technikou částečné dynamické rekonfigurace použitou pro opravu specifické oblasti v konfigurační paměti FPGA odpovídající modulu TMR, ve kterém byla detekována porucha. Obvody FPGA s konfigurační pamětí typu SRAM jsou citlivé na vznik poruch typu SEU. Tyto poruchy se mohou objevit jak v konfigurační paměti FPGA, tak v aplikační logice implementovaného systému. Synchronizace stavu (nebo jiná alternativní technika pro opravu stavu systému) je nezbytná pro zachování provozuschopnosti a vysoké dostupnosti systému odolného proti poruchám, i když lze díky rekonfiguraci FPGA opravit poruchy typu SEU uvnitř konfigurační paměti. Synchronizace stavu využívá redundantních informací o stavu systému vycházejících z architektury TMR pro opravu poruchy typu SEU. Tyto poruchy mohou vzniknout přímo v sekvenčních registrech aplikační logiky nebo jako důsledek selhání funkce implementované logiky ovlivněné poruchou v konfigurační paměti FPGA. Návrhový proces pro návrh opravovaného systému s využitím technik implementace architektury TMR a mechanismu rekonfigurace, který byl využit i při implementaci experimentálních systému prezentovaných v této disertační práci, byl popsán v kapitole 5.

Jádrem disertační práce jsou kapitoly 6, 7 a 8. V kapitole 6 byla popsán teoretický rámec navržené metodiky zahrnující: úvod do aplikace metodiky, představení základních principů návrhu metod synchronizace stavu, popis funkce logiky synchronizačních obvodů, definici dynamických a statických parametrů metod synchronizace stavu a popis průběhu uvažovaného procesu opravy stavu systému po vzniku a detekci poruchy. Tato kapitola blíže popisuje způsob výběru vhodné metody synchronizace vzhledem ke specifickým vlastnos-

tem cílového systému (typ architektury TMR) a závislosti provádění funkce systému na udržení informací o předchozích stavech. Navržená metodika rozlišuje mezi třemi úrovněmi implementace synchronizace stavu – synchronizací systému ve společném stavu, synchronizací na úrovni redundantních modulů a synchronizaci na úrovni RTL. Pro každou z těchto úrovní synchronizace zde byly popsány způsoby návrhu propojení hardwarových jednotek a implementace synchronizace stavu. V kapitolách 7 a 8 jsou popsány dvě případové studie zaměřené na návrh metod synchronizace stavu podle popsané metodiky pro systém řadiče sběrnice CAN řízený stavovými automaty a pro systém řízený mikrokontrolerem s procesorovým jádrem NEO430. Na těchto dvou rozličných číslicových systémech byl demonstrován způsob zabezpečení systému s využitím architektury TMR, ověřena technika opravy přechodných poruch typu SEU s využitím částečné dynamické rekonfigurace a předveden návrh specifických metod synchronizace stavu s využitím popsané metodiky.

Během vlastního výzkumu byly publikovány tři vědecké články na mezinárodních konferencích [90] [93] [94], které se zabývaly návrhem systému řadiče sběrnice CAN odolného proti poruchám a mechanismy opravy stavu systému s využitím rekonfigurace a synchronizace. Výsledky těchto výzkumných aktivit byly popsány v kapitole 7. Článek [94] definoval základní opěrné body navržené metodiky, blíže prezentované v kapitole 6. Tento příspěvek obdržel na konferenci IEEE cenu **„Best Paper Award“**.

Další výzkumné aktivity byly zaměřeny na návrh metod synchronizace pro systémy řízené procesorem. Tento výzkum byl popsán v kapitole 8. Výsledky výzkumu byly publikovány ve dvou vědeckých článcích [91] a [92] na mezinárodních konferencích. Navržená metodika byla díky znalostem získaných při návrhu a implementaci zabezpečení pro experimentální systém řízený mikrokontrolerem NEO430 dále rozšířena do finální podoby.

Návrh metodiky pro návrh synchronizace stavu tématicky doplňuje řadu disertačních prací [87] [65] [22] vážených kolegů z univerzit v České republice, kteří se zabývali technikami využití PDR pro opravu přechodných a trvalých poruch v obvodech FPGA s konfigurační pamětí SRAM.

9.2 Přínos práce

Hlavním přínosem této disertační práce je vytvoření metodiky pro návrh synchronizace stavu rekonfigurovatelných modulů v rámci zabezpečené architektury CG-TMR. Navržená metodika spolu s výsledky výzkumné práce a provedených experimentů, které byly implementovány během řešení výzkumných otázek a cílů disertační práce popsaných v kapitole 4, přináší následující nové poznatky:

- Byly prozkoumány možnosti návrhu číslicových systémů jako systémů odolných proti poruchám do obvodů FPGA se zaměřením na využití obvodové redundance, schopnosti částečné dynamické rekonfigurace obvodů FPGA a mechanismy opravy a zotavení stavu systému z poruchy.
- Byla popsána problematika implementace TMR na různých úrovních návrhu, tedy hrubozrnná a jemnozrnná architektura TMR (označované v této práci jako CG-TMR a FG-TMR). Součástí experimentů bylo vytvoření návrhu mikrokontroleru zabezpečeného pomocí architektury FG-TMR na úrovni RTL, která byla automaticky implementována nástrojem BL-TMR. Výsledkem porovnání spolehlivosti implementace CG-TMR (s nebo bez opravy) s implementaci FG-TMR je, že FG-TMR přináší značné zlepšení v odolnosti systému proti poruchám na úkor dramatického snížení pracovní frekvence systému.

- Byla vytvořena metodika pro návrh a implementaci číslicových obvodů pro synchronizaci stavu rekonfigurovatelných modulů TMR architektury systému odolného proti poruchám implementovaného do FPGA. Navržená metodika doplňuje řešení problematiky návrhu rekonfigurovatelného systému zabezpečeného pomocí architektury TMR. Tento způsob návrhu rekonfigurovatelného systému byl aplikován i při implementaci experimentálních systémů řadiče sběrnice CAN a mikrokontroleru s jádrem procesoru NEO430.

- Na dvou typických číslicových systémech byl demonstrován návrh rekonfigurovatelného systému a využití vlastní metodiky pro návrh synchronizace stavu. Pro systém řadiče sběrnice CAN, jakožto zástupce systému řízeného stavovými automaty, byly navrženy a implementovány metody provádějící synchronizaci stavu pomocí sériového a paralelního propojení registrů. Pro systém mikrokontroleru, jakožto zástupce systému řízeného procesorem, byly navrženy a implementovány metody provádějící synchronizaci stavu pomocí synchronizačního restartu a sdílené datové paměti. Pro realizaci synchronizace stavu byly implementovány specifické obvody (řadič synchronizace, arbitr synchronizace) a modifikována architektura daných systémů.

- Při návrhu zabezpečení systému mikrokontroleru byla navržena a implementována architektura ReSyTMR, která umožňuje zabezpečenému systému vlastní řízení procesu opravy a zotavení se z poruchy. Kromě zabezpečeného mikrokontroleru je součásti každého modulu architektury CG-TMR jednotka monitoru stavu odolného systému (označována v této práci jako FTM). FTM umožňuje řídit provádění jednotlivých fází procesu obnovy stavu systému z poruchy na základě informací ze ztrojených komparátorů výstupných signálů modulů TMR a aktuálního stavu celého systému.

- Pro ověření správné funkce navržených mechanismů opravy stavu systému s využitím rekonfigurace a synchronizace stavu bylo implementováno verifikační prostředí PDR Framework. Jednotka PDR Framework tvoří prostředníka umožňujícího komunikaci mezi experimentálním PC (návrhářem), zabezpečeným systémem a číslicovými obvody zodpovědnými za opravu stavu systému. PDR Framework implementuje jak uživatelský terminál, užitečný pro samotného návrháře, tak speciální sériový komunikační protokol vhodný pro provádění automatický řízených experimentů. Obě komunikační rozhraní umožňují provádět různé příkazy pro řízení a monitorování experimentů, tedy procesu rekonfigurace a synchronizace stavu.

- Při experimentálním ověření bylo porovnáno několik variant zabezpečeného systému mikrokontroleru. S využitím nástroje pro injekci poruch typu SEU do konfigurační paměti FPGA byly postupně ověřeny ukazatele spolehlivosti nezabezpečeného návrhu, návrhu zabezpečeného pomocí CG-TMR, návrhu zabezpečeného pomocí FG-TMR a návrhu zabezpečeného pomocí ReSyTMR.

Zabezpečené systémy řadiče sběrnice CAN a mikrokontroleru NEO430 byly implementovány pro obvod FPGA Virtex-5 od firmy Xilinx. Nicméně, samotná aplikovatelnost metodiky pro návrh metod synchronizace není nijak závislá na typu obvodu FPGA. Částečná dynamická rekonfigurace je podporována množstvím obvodů, jak od firmy Xilinx tak např. od firmy Intel. Technika synchronizace stavu může být využita jako prostředek pro opravu stavu paměťových prvků v aplikační vrstvě FPGA i v případě, že zabezpečený systém nevyužívá rekonfiguraci pro opravu přechodných poruch v konfigurační paměti.

9.3 Možné rozšíření práce

Tato disertační práce nepostihla všechna témata, která se k řešenému problému vážou. Během výzkumu se objevili další výzkumné otázky , které by bylo zajímavé prozkoumat, a oblasti, kam by další výzkum navazující na tuto disertační práci mohl směřovat. Možná rozšíření této disertační práce jsou popsány v následujících podkapitolách.

9.3.1 Synchronizace stavu při opravě trvalé poruchy pomocí relokace

Synchronizace stavu odolného systému může být potřebná také v případě opravy trvalé poruchy s využitím techniky relokace, kdy je v rámci rekonfigurovatelné architektury TMR nutné přesunout logiku jednoho redundantního modulu na jiné místo v konfigurační paměti FPGA. Základní principy techniky opravy stavu systému v případě výskytu trvalé poruchy v konfigurační paměti FPGA byly popsány v kapitole 2.6.2. Při opravě se využívá konfiguračních bitstreamů, které jsou předkompilovány nebo vytvořeny za běhu systému. Návrh rekonfigurovatelného systému, který umožňuje relokaci částečných bitstreamů vyžaduje dodatečné kroky pro zajištění fixního rozhraní mezi rekonfigurovatelnými moduly a statickou logikou. Rozdíly v technice návrhu byly popsány v kapitole 5.2.

Způsob řešení návrhu synchronizace stavu pro systém využívající relokaci se neliší od metodiky popsané v této práci. Nicméně výsledkem takového výzkumu by byl systém odolný proti poruchám s velmi vysokou dostupností, která by byla dosažena díky schopnosti opravy systému z přechodných i trvalých poruch za běhu systému.

9.3.2 Synchronizace stavu pomocí vyčtení stavu registrů z FPGA

Výzkum prezentovaný v této disertační práce byl zaměřen na metody synchronizace stavu využívající fyzické propojení mezi redundantními moduly. Nicméně alternativním řešením celého problému synchronizace stavu je využití schopnosti částečné dynamické rekonfigurace obvodů FPGA ke zpětnému vyčtení konfigurační paměti včetně stavu implementovaných funkčních obvodů. Tímto způsobem může být opravovaný modul architektury TMR synchronizován za pomocí stavu extrahovaného z jiného modulu, který pracuje správně.

Technika zpětného vyčítání konfigurační paměti se využívá v praxi pro verifikaci správně nakonfigurované konfigurační paměti FPGA. Kromě toho lze při zpětném vyčítání vyčíst i obsah registrů sekvenční logiky [22]. K tomu je zapotřebí využití komponenty CAPTURE_VIRTEX5 nebo přímého zápisu příkazu GCAPTURE do konfiguračního registru obvodu FPGA [111]. Vyčítání stavu může být průběžné, kdy dochází k aktualizaci ukládaného stavu vnitřních registrů na každou náběžnou hranu hodinového signálu. Případně může být vyčtena hodnota stavu vnitřních registrů jednorázově. Vlastní vyčítání je pak realizováno za pomocí některého z konfiguračních rozhraní obvodu FPGA. Po dokončení zpětného vyčtení konfigurační paměti je nutné provést přiřazení mezi hodnotami vyčtených bitů a jejich odpovídajícím umístění v implementovaném číslicovém návrhu. K tomu je nutné vygenerovat tzv. alokační soubor logiky, ve kterém jsou uloženy informace o umístění jednotlivých registrů v obvodě FPGA a spojistosti mezi jednotlivými bity konfiguračních rámců a vnitřních hodnot registrů. Následně je nutné provést opětovný zápis hodnot registrů do rekonfigurovatelného modulu, který má být synchronizován [26]. Pro to je nutné vytvořit konfigurační bitstream složený z původních konfigurčních dat a extrahovaných hodnot registrů. Pro zápis konfigurace s inicializačními hodnotami synchronizovaných registrů lze využít komponentu STARTUP_VIRTEX5 nebo příkaz GRESTORE pro konfigurační registr FPGA [111].

Tato technika synchronizace stavu vyžaduje detailní znalost struktury konfigurační paměti a formátu konfiguračního bitstreamu pro daný typ obvodu FPGA. Nespornou výhodou použití této techniky je, že není nutné modifikovat architekturu cílového systému ani implementovat synchronizační propojení mezi redundantními moduly architektury TMR.

Literatura

[1] Adetomi, A.; Enemali, G.; Iturbe, X.; aj.: R3TOS-Based Integrated Modular Space Avionics for On-Board Real-Time Data Processing. *2018 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2018*, 2018: s. 1–8, doi:10.1109/AHS.2018.8541369.

[2] Agiakatsikas, D.: *High-level synthesis of triple modular redundant FPGA circuits with energy efficient error recovery mechanisms*. Dizertační práce, UNSW Sydney, 2019.

[3] Amano, H.: *Design and Structures of FPGAs*. Springer, 2018, ISBN 9789811308239.

[4] Andina, J. J. R.; de la Torre Arnanz, E.; Peña, M. D. V.: *FPGAs: Fundamentals, advanced features, and applications in industrial electronics*. CRC Press, 2017, ISBN 9781482282290, 1–250 s., doi:10.1201/9781315162133.

[5] Asadi, G.; Tahoori, M. B.: Soft error rate estimation and mitigation for SRAM-based FPGAs. *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA*, 2005: s. 149–160, doi:10.1145/1046192.1046212.

[6] Astarloa, A.; Lazaro, J.; Bidarte, U.; aj.: An autonomous fault tolerant system for CAN communications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, ročník 6098 LNAI, č. PART 3, 2010: s. 281–290, ISSN 03029743, doi:10.1007/978-3-642-13033-5_29.

[7] Avižienis, A.; Laprie, J. C.; Randell, B.; aj.: Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, ročník 1, č. 1, 2004: s. 11–33, ISSN 15455971, doi:10.1109/TDSC.2004.2.

[8] Azambuja, J. R.; Pilotto, C.; Kastensmidt, F. L.: Mitigating soft errors in SRAM-based FPGAs by using large grain TMR with selective partial reconfiguration. *Proceedings of the European Conference on Radiation and its Effects on Components and Systems, RADECS*, Září 2008: s. 288–293, ISSN 0379-6566, doi:10.1109/RADECS.2008.5782729.

[9] Azambuja, J. R.; Sousa, F.; Rosa, L.; aj.: Evaluating large grain TMR and selective partial reconfiguration for soft error mitigation in SRAM based FPGAs. *2009 15th IEEE International On-Line Testing Symposium, IOLTS 2009*, 2009: s. 101–106, doi:10.1109/IOLTS.2009.5195990.

[10] Baumann, R.; Kruckmeyer, K.: *Radiation handbook for electronics*. Texas Instruments, 2019.

[11] Baumann, R. C.: Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, ročník 5, č. 3, 2005: s. 305–315, ISSN 15304388, doi:10.1109/TDMR.2005.853449.

[12] Benites, L. A. C.; Kastensmidt, F. L.: Automated design flow for applying Triple Modular Redundancy (TMR) in complex digital circuits. *2018 IEEE 19th Latin-American Test Symposium, LATS 2018*, 2018: s. 1–4, doi:10.1109/LATW.2018.8349668.

[13] Berg, M.; Poivey, C.; Petrick, D.; aj.: Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis. *IEEE Transactions on Nuclear Science*, ročník 55, č. 4, 2008: s. 2259–2266, ISSN 00189499, doi:10.1109/TNS.2008.2001422.

[14] Bolchini, C.; Miele, A.; Sandionigi, C.: A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs. *IEEE Transactions on Computers*, ročník 60, č. 12, 2011: s. 1744–1758, ISSN 00189340, doi:10.1109/TC.2010.281.

[15] Bolchini, C.; Miele, A.; Sandionigi, C.: Autonomous fault-tolerant systems onto SRAM-based FPGA platforms. *Journal of Electronic Testing: Theory and Applications (JETTA)*, ročník 29, č. 6, 2013: s. 779–793, ISSN 09238174, doi:10.1007/s10836-013-5418-4.

[16] Bozzoli, L.; Sterpone, L.: Self rerouting of dynamically reconfigurable SRAM-based FPGAS. *2017 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2017*, 2017: s. 77–84, doi:10.1109/AHS.2017.8046362.

[17] Bridgford, B.; Carmichael, C.; Tseng, C. W.: Single-Event Upset Mitigation Selection Guide. URL: <www.xilinx.com/support/documentation/application_notes/xapp987.pdf>, [cit. 2020-08-30], (XAPP987).

[18] Carrol, James F: BYU-LANL Triple Modular Redundancy Usage Guide. URL: <http://sourceforge.net/projects/byuediftools/files/byuediftools>, [cit. 2020-08-30].

[19] Courtland, R.: Intel Now Packs 100 Million Transistors in Each Square Millimeter. URL: <https://spectrum.ieee.org/nanoclast/semiconductors/processors/intel-now-packs-100-million-transistors-in-each-square-millimeter>, [cit. 2020-04-04].

[20] Cronquist, B.; Sarpa, M.; Wang, J. J.; aj.: Modifications of COTS FPGA Devices for Space Applications. *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, 1998.

[21] Di Carlo, S.; Gambardella, G.; Prinetto, P.; aj.: A novel methodology to increase fault tolerance in autonomous FPGA-based systems. *Proceedings of the 2014 IEEE 20th International On-Line Testing Symposium, IOLTS 2014*, 2014: s. 87–92, doi:10.1109/IOLTS.2014.6873677.

[22] Drahoňovský, Tomáš: *Rekonfigurovatelný systém na FPGA obvodu*. Dizertační práce, Technická univerzita v Liberci, Liberec, 2014.

[23] Dubrova, E.: *Fault-Tolerant Design*. Springer, 2013, ISBN 9781461421122, doi:10.1007/978-1-4614-2113-9.

[24] Ebrahimi, M.; Miremadi, S. G.; Asadi, H.: ScTMR: A scan chain-based error recovery technique for TMR systems in safety-critical applications. In *2011 Design, Automation Test in Europe*, 2011, s. 1–4.

[25] Ebrahimi, M.; Miremadi, S. G.; Asadi, H.; aj.: Low-cost scan-chain-based technique to recover multiple errors in TMR systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, ročník 21, č. 8, 2013: s. 1454–1468, ISSN 10638210, doi:10.1109/TVLSI.2012.2213102.

[26] Eckert, M.; Meyer, D.; Klauer, B.: Context Save and Restore of Partial Reconfiguration Regions for Xilinx FPGAs. In *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, IEEE, 07 2019, ISBN 9781728147703, s. 5–12, doi:10.1109/recosoc48741.2019.9034942.

[27] ECSS: Space product assurance - Techniques for radiation effects mitigation in ASICs and FPGAs handbook (ECSS-Q-HB-60-02A). Září 2016.

[28] Einspruch, N.: *Application Specific Integrated Circuit (ASIC) Technology*, ročník 23. Elsevier, 1991, ISBN 9780122341236, doi:10.1016/B978-0-122-34123-6.X5001-X.

[29] Faggin, F.; Hoff, M. E.; Mazor, S.; aj.: The history of the 4004. *IEEE Micro*, ročník 16, č. 6, 1996: s. 10–20.

[30] Fiethe, B.; Bubenhagen, F.; Lange, T.; aj.: Adaptive hardware by dynamic reconfiguration for the solar orbiter PHI instrument. *Proceedings of the 2012 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2012*, 2012: s. 31–37, doi:10.1109/AHS.2012.6268666.

[31] Foster, H.: 2018 FPGA Functional Verification Trends. *Proceedings - 2018 19th International Workshop on Microprocessor and SOC Test, Security and Verification, MTV 2018*, 2018: s. 40–45, doi:10.1109/MTV.2018.00018.

[32] Fujino, M.; Tanaka, H.; Ichinomiya, Y.; aj.: Fault Recovery Technique for TMR Softcore Processor System Using Partial Reconfiguration. In *Algorithms and Architectures for Parallel Processing*, editace Y. Xiang; I. Stojmenovic; B. O. Apduhan; G. Wang; K. Nakano; A. Zomaya, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-33078-0, s. 392–404.

[33] Fulton, R.; Vandermolen, R.: *Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254*. USA: CRC Press, 2014, ISBN 1482206056.

[34] Gong, L.; Kroh, A.; Agiakatsikas, D.; aj.: Reliable SEU monitoring and recovery using a programmable configuration controller. *2017 27th International Conference on Field Programmable Logic and Applications, FPL 2017*, 2017, doi:10.23919/FPL.2017.8056798.

[35] Gong, L.; Wu, T.; Nguyen, N. T.; aj.: A programmable configuration controller for fault-tolerant applications. *Proceedings of the 2016 International Conference on Field-Programmable Technology, FPT 2016*, 2017: s. 117–124, doi:10.1109/FPT.2016.7929515.

[36] Hauck, S.; DeHon, A.: *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, ISBN 9780080556017.

[37] Herrera-Alzu, I.; López-Vallejo, M.: Design techniques for Xilinx Virtex FPGA configuration memory scrubbers. *IEEE Transactions on Nuclear Science*, ročník 60, č. 1, 2013: s. 376–385, ISSN 00189499.

[38] Hlavička, G. a. B., Racek: *Číslicové systémy odolné proti poruchám*. Praha: Vydavatelství ČVUT, 1992, ISBN 80-01-00852-5.

[39] Hogan, J. A.; Weber, R. J.; Lameres, B. J.: A network-on-chip for radiation tolerant, multi-core FPGA systems. *IEEE Aerospace Conference Proceedings*, 2014, ISSN 1095323X, doi:10.1109/AERO.2014.6836322.

[40] Hong, C.; Benkrid, K.; Ebrahim, A.; aj.: Virtual shared memory architecture for inter-task communication in partial reconfigurable systems. *Proceedings of the International Conference on Microelectronics, ICM*, 2012: s. 3–6, doi:10.1109/ICM.2012.6471361.

[41] Hsiung, P.-A.; Santambrogio, M.; Huang, C.-H.: *Reconfigurable System Design and Verification*. CRC Press, 2009, doi:10.1201/9781420062670.

[42] Ichinomiya, Y.; Amagasaki, M.; Iida, M.; aj.: Improving the Soft-error Tolerability of a Soft-core Processor on an FPGA using Triple Modular Redundancy and Partial Reconfiguration. *Journal of Next Generation Information Technology*, ročník 2, č. 3, 2011: s. 35–48, ISSN 2092-8637, doi:10.4156/jnit.vol2.issue3.3.

[43] Ichinomiya, Y.; Tanoue, S.; Amagasaki, M.; aj.: Improving the robustness of a softcore processor against SEUs by using TMR and partial reconfiguration. *Proceedings - IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2010*, 2010: s. 47–54, doi:10.1109/FCCM.2010.16.

[44] Ichinomiya, Y.; Usagawa, S.; Amagasaki, M.; aj.: Designing flexible reconfigurable regions to relocate partial bitstreams. *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012*, 2012: str. 241, doi:10.1109/FCCM.2012.51.

[45] Iturbe, X.; Benkrid, K.; Arslan, T.; aj.: Methods and mechanisms for hardware multitasking: Executing and synchronizing fully relocatable hardware tasks in xilinx FPGAs. *Proceedings - 21st International Conference on Field Programmable Logic and Applications, FPL 2011*, 2011: s. 295–300, doi:10.1109/FPL.2011.60.

[46] Iturbe, X.; Benkrid, K.; Hong, C.; aj.: R3TOS: A novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on FPGAs. *IEEE Transactions on Computers*, ročník 62, č. 8, 2013: s. 1542–1556, ISSN 00189340, doi:10.1109/TC.2013.79.

[47] Johnson, J. M.; Wirthlin, M. J.: *Voter insertion algorithms for FPGA designs using triple modular redundancy.* Dizertační práce, 2010, doi:10.1145/1723112.1723154.

[48] Kanekawa, N.; Eishi, I.; Suga, T.; aj.: *Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances.* Springer-Verlag New York, Leden 2011, 1-204 s., doi:10.1007/978-1-4419-6715-2.

[49] Kastensmidt, F.; Rech, P.: *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design.* Springer, 2015, ISBN 9783319143521, 3–17 s., doi:10.1007/978-3-319-14352-1_1.

[50] Kizheppatt, V.: *Design Automation for Partially Reconfigurable Adaptive Systems.* Dizertační práce, Leden 2015.

[51] Koch, D.: *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications (Lecture Notes in Electrical Engineering).* Springer, 2012, ISBN 1461412242, 317 s.

[52] Kopetz, H.: *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Springer, druhé vydání, 2011, ISBN 0792398947, 338 s., doi:10.1007/978-1-4419-8237-7.

[53] Kretzschmar, U.; Astarloa, A.; Lazaro, J.; aj.: Robustness of different TMR granularities in shared wishbone architectures on SRAM FPGA. *2012 International Conference on Reconfigurable Computing and FPGAs*, 2012, doi:10.1109/ReConFig.2012.6416785.

[54] Kretzschmar, U.; Gomez-Cornejo, J.; Astarloa, A.; aj.: Synchronization of faulty processors in coarse-grained TMR protected partially reconfigurable FPGA designs. *Reliability Engineering and System Safety*, ročník 151, 2016: s. 1–9, ISSN 09518320, doi:10.1016/j.ress.2015.12.018.

[55] Kulis, S.: Single Event Effects mitigation with TMRG tool. *Journal of Instrumentation*, ročník 12, č. 1, 2017: s. 0–6, ISSN 17480221, doi:10.1088/1748-0221/12/01/C01082.

[56] Kuon, I.; Tessier, R.; Rose, J.: FPGA Architecture: Survey and Challenges. *Foundations and Trends in Electronic Design Automation*, ročník 2, č. 2, 2008: s. 135–253, ISSN 1551-3939, doi:10.1561/1000000005.

[57] Lach, J.; Mangione-Smith, W. H.; Potkonjak, M.: Low overhead fault-tolerant FPGA systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, ročník 6, č. 2, 1998: s. 212–221, ISSN 10638210, doi:10.1109/92.678870.

[58] Lach, J.; Mangione-Smith, W. H.; Potkonjak, M.: Algorithms for efficient runtime fault recovery on diverse FPGA architectures. *Proceedings - 1999 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT 1999*, 1999: s. 386–394, doi:10.1109/DFTVS.1999.802906.

[59] Lavin, C.; Padilla, M.; Lamprecht, J.; aj.: RapidSmith: Do-it-yourself CAD tools for Xilinx FPGAs. In *Proceedings - 21st International Conference on Field Programmable Logic and Applications, FPL 2011*, Xdl, 2011, ISBN 9780769545295, s. 349–355, doi:10.1109/FPL.2011.69.

[60] Lee, G.; Agiakatsikas, D.; Wu, T.; aj.: TLegUp: A TMR code generation tool for SRAM-based FPGA applications using HLS. *Proceedings - IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2017*, , č. iii, 2017: s. 129–132, doi:10.1109/FCCM.2017.57.

[61] Liu, S.; Pittman, R. N.; Forin, A.; aj.: Achieving energy efficiency through runtime partial reconfiguration on reconfigurable systems. *Transactions on Embedded Computing Systems*, ročník 12, č. 3, 2013: s. 265–272, ISSN 15399087, doi:10.1145/2442116.2442122.

[62] May, T. C.; Woods, M. H.: A New Physical Mechanism for Soft Errors in Dynamic Memories. In *16th International Reliability Physics Symposium*, 1978, s. 33–40.

[63] Mentor Graphics: Precision Hi-Rel Advanced FPGA Synthesis. URL: <https://s3.amazonaws.com/s3.mentor.com/public_documents/datasheet/products/fpga/precision-hirel-ds.pdf>, [cit. 2020-07-05].

[64] Microchip Technology Inc.: MCP2515 - Stand-Alone CAN Controller with SPI Interface. Září 2016.

[65] Miculka, L.: *Methodology for fault tolerant systems design into limited implementation area in FPGA*. Dizertační práce, FIT VUT v Brně, Brno, 2017.

[66] Morillo, A.; Astarloa, A.; Lazaro, J.; aj.: Reliable microprocessors for FPGAs: State of the art and trends. *Applied Electronics (AE), 2010 International Conference on*, 2010, ISSN 1803-7232.

[67] Morillo, A.; Astarloa, A.; Lazaro, J.; aj.: Known-blocking. Synchronization method for reliable processor using TMR amp; DPR in SRAM FPGAs. In *Programmable Logic (SPL), 2011 VII Southern Conference on*, Duben 2011, s. 57–62.

[68] Mousavi, M.; De, S.; Pourshaghaghi, H. R.; aj.: Fault Tolerant FPGAs : Where to Spend the Effort ? *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019: s. 651–654, doi:10.1109/DSD.2019.00103.

[69] von Neumann, J.: Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. *Automata Studies*, 1956: s. 43–98.

[70] Nicolaidis, M.: *Soft Errors in Modern Electronic Systems*. Springer, 2011, ISBN 978-1-4419-6992-7.

[71] Niknahad, M.: *Using Fine Grain Approaches for highly reliable Design of FPGA-based Systems in Space*. Dizertační práce, Karlsruher Institut für Technologie, Karlsruhe, 2012.

[72] Nolting, S.: The NEO430 Processor. URL: <github.com/stnolting/neo430>, [cit. 2020-02-29].

[73] Pedroni, V.: *Digital Electronics and Design with VHDL*. Morgan Kaufmann Publishers Inc., 2008, ISBN 0123742706, 694 s.

[74] Pham, H.; Pillement, S.; Demigny, D.: Evaluation of Fault-Mitigation Schemes for Fault-Tolerant Dynamic MPSoC. In *2010 International Conference on Field Programmable Logic and Applications*, Srpen 2010, ISSN 1946-1488, s. 159–162, doi:10.1109/FPL.2010.38.

[75] Pham, H. M.; Pillement, S.; Demigny, D.: A fault-tolerant layer for dynamically reconfigurable multi-processor system-on-chip. *ReConFig'09 - 2009 International Conference on ReConFigurable Computing and FPGAs*, 2009: s. 284–289, doi:10.1109/ReConFig.2009.47.

[76] Pham, H. M.; Pillement, S.; Piestrak, S. J.: Low-overhead fault-tolerance technique for a dynamically reconfigurable softcore processor. *IEEE Transactions on Computers*, ročník 62, č. 6, 2013: s. 1179–1192, ISSN 00189340, doi:10.1109/TC.2012.55.

[77] Pilotto, C.; Azambuja, J. R.; Kastensmidt, F. L.: Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications. *Proceedings of the 21st annual symposium on Integrated circuits and system design*, 2008: s. 199–204, doi:10.1145/1404371.1404426.

[78] Pratt, B.; Caffrey, M.; Graham, P.; aj.: Improving FPGA design robustness with partial TMR. *IEEE International Reliability Physics Symposium Proceedings*, 2006: s. 226–232, ISSN 15417026.

[79] Robert Bosch GmbH: CAN Specification 2.0. 1991.

[80] Samudrala, P. K.; Ramos, J.: Selective Triple Modular Redundancy (STMR) Based Single-Event Upset (SEU) Tolerant Synthesis for FPGAs. ročník 51, č. 5, 2004: s. 2957–2969.

[81] Schütz, M.; Steininger, A.; Huemer, F.; aj.: State Recovery for Coarse-Grain TMR Designs in FPGAs Using Partial Reconfiguration. *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2018: s. 1–6.

[82] Siegle, F.; Vladimirova, T.; Ilstad, J.; aj.: Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications. *ACM Computing Surveys*, ročník 47, č. 2, 2015: s. 1–34, ISSN 03600300, doi:10.1145/2671181.

[83] Smith, G. L.; De La Torre, L.: Techniques to enable FPGA based reconfigurable fault tolerant space computing. *IEEE Aerospace Conference Proceedings*, ročník 2006, 2006: str. 11 pp., ISSN 1095323X, doi:10.1109/aero.2006.1655958.

[84] Spitzer, C. R.; Ferell, U.; Ferrel, T.: *Digital Avionics Handbook*. Electrical engineering handbook series, Boca Raton, FL, USA: CRC Press, třetí vydání, 2015, ISBN 978-1-4398-6861-4.

[85] Stapelberg, R.: *Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design*. Springer, 2009, ISBN 9781848001756.

[86] Stock Flight Systems: CANAerospace - Interface specification for airborne CAN applications v1.7. 2006.

[87] Straka, M.: *Metodologie pro navrh cislicovych obvodu se zvysenou spolehlivosti*. Dizertační práce, FIT VUT v Brně, Brno, 2013.

[88] Straka, M.; Kastil, J.; Kotasek, Z.: SEU simulation framework for Xilinx FPGA: First step towards testing fault tolerant systems. *Proceedings - 2011 14th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2011*, 2011: s. 223–230, doi:10.1109/DSD.2011.32.

[89] Szurman, K.: *Využití moderních metod zvyšování spolehlivosti pro implementaci řídicího systému*. Diplomová práce, FIT VUT v Brně, Brno, 2012, vedoucí práce Ing. Jan Kaštil, PhD.

[90] Szurman, K.; Kaštil, J.; Straka, M.; aj.: Fault Tolerant CAN Bus Control System Implemented into FPGA. In *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013*, 2013, ISBN 978-1-4673-6136-1, s. 289–292, doi:10.1109/DDECS.2013.6549837.

[91] Szurman, K.; Kotásek, Z.: Coarse-Grained TMR Soft-Core Processor Fault Tolerance Methods and State Synchronization for Run-Time Fault Recovery. In *20th IEEE Latin American Test Symposium (LATS 2019)*, 2019, ISBN 978-1-7281-1756-0, s. 32–35, doi:10.1109/LATW.2019.8704639.

[92] Szurman, K.; Kotásek, Z.: Run-Time Reconfigurable Fault Tolerant Architecture for Soft-Core Processor neo430. In *22nd International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2019)*, 2019, ISBN 978-1-7281-0073-9, s. 136–140, doi:10.1109/DDECS.2019.8724636.

[93] Szurman, K.; Mičulka, L.; Kotásek, Z.: State Synchronization after Partial Reconfiguration of Fault Tolerant CAN Bus Control System. In *17th Euromicro Conference on Digital Systems Design*, 2014, ISBN 978-1-4799-5793-4, s. 704–707, doi:10.1109/DSD.2014.103.

[94] Szurman, K.; Mičulka, L.; Kotásek, Z.: Towards a State Synchronization Methodology for Recovery Process after Partial Reconfiguration of Fault Tolerant Systems. In *9th IEEE International Conference on Computer Engineering and Systems*, 2014, ISBN 978-1-4799-6594-6, s. 231–236, doi:10.1109/ICCES.2014.7030963.

[95] Tambara, L. A.; Almeida, F.; Rech, P.; aj.: Measuring Failure Probability of Coarse and Fine Grain TMR Schemes in SRAM-based FPGAs Under Neutron-Induced Effects. In *Applied Reconfigurable Computing*, editace K. Sano; D. Soudris; M. Hübner; P. C. Diniz, Cham: Springer International Publishing, 2015, ISBN 978-3-319-16214-0, s. 331–338.

[96] Tanoue, S.; Ishida, T.; Ichinomiya, Y.; aj.: A NOVEL STATES RECOVERY TECHNIQUE FOR THE TMR SOFTCORE PROCESSOR. *Fpl*, 2009: s. 543–546.

[97] Trimberger, S. M.: Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology. *Proceedings of the IEEE*, ročník 103, č. 3, 2015: s. 318–331, ISSN 00189219.

[98] Urban, M.; Nentvich, O.; Stehlikova, V.; aj.: VZLUSAT-1: Nanosatellite with miniature lobster eye X-ray telescope and qualification of the radiation shielding composite for space application. *Acta Astronautica*, ročník 140, č. September 2016, 2017: s. 96–104, ISSN 00945765, doi:10.1016/j.actaastro.2017.08.004.

[99] Villalta, I.; Bidarte, U.; Gomez-Cornejo, J.; aj.: Dependability in FPGAs, a Review. *2015 Conference on Design of Circuits and Integrated Systems, DCIS 2015*, 2016, doi:10.1109/DCIS.2015.7388570.

[100] Wakerly, J. F.: Transient Failures in Triple Modular Redundancy Systems with Sequential Modules. *IEEE Transactions on Computers*, ročník C-24, č. 5, 1975: s. 570–573, ISSN 00189340, doi:10.1109/T-C.1975.224263.

[101] Wakerly, J. F.: Synchronization and Matching in Redundant Systems. *IEEE Transactions on Computers*, ročník C-27, č. 6, 1978: s. 531–539, ISSN 00189340, doi:10.1109/TC.1978.1675144.

[102] Weste, N.; Harris, D.: *CMOS VLSI Design: A Circuits and Systems Perspective*. USA: Addison-Wesley Publishing Company, Čtvrté vydání, 2010, ISBN 0321547748.

[103] Wirthlin, M.: High-reliability FPGA-based systems: Space, high-energy physics, and beyond. *Proceedings of the IEEE*, ročník 103, č. 3, 2015: s. 379–389, ISSN 15582256, doi:10.1109/JPROC.2015.2404212.

[104] Xilinx Inc.: Constraints Guide. URL: <www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/cgd.pdf>, [cit. 2020-08-30], (UG625).

[105] Xilinx Inc.: Difference-Based Partial Reconfiguration. URL: <www.xilinx.com/support/documentation/application_notes/xapp290.pdf">, [cit. 2020-08-30], (XAPP290).

[106] Xilinx Inc.: Partial Reconfiguration User Guide. URL: <www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug702.pdf>, [cit. 2020-08-30], (UG702).

[107] Xilinx Inc.: PicoBlaze 8-bit Embedded Microcontroller User Guide. URL: <www.xilinx.com/support/documentation/ip_documentation/ug129.pdf>, [cit. 2020-08-30], (UG129).

[108] Xilinx Inc.: PlanAhead User Guide. URL: <www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/PlanAhead_UserGuide.pdf>, [cit. 2020-08-30], (UG632).

[109] Xilinx Inc.: Versal: The First Adaptive Compute Acceleration Platform (ACAP). URL: <www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf>, Září [cit. 2020-08-30], (WP505).

[110] Xilinx Inc.: Virtex-5 Family Overview: Product Specification. URL: <www.xilinx.com/support/documentation/data_sheets/ds100.pdf>, [cit. 2020-08-30], (DS100).

[111] Xilinx Inc.: Virtex-5 FPGA: Configuration User Guide. URL: <www.xilinx.com/support/documentation/user_guides/ug191.pdf>, [cit. 2020-08-30], (UG191).

[112] Xilinx Inc.: XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices. URL: <www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/xst.pdf>, [cit. 2020-08-30], (UG627).

[113] Xiu, L.: Time Moore: Exploiting Moore's Law from the Perspective of Time. *IEEE Solid-State Circuits Magazine*, ročník 11, č. 1, 2019: s. 39–55, ISSN 19430590, doi:10.1109/MSSC.2018.2882285.

[114] Yermalayeva, D.: *Vliv radiace na vlastnosti polovodičových součástek*. Diplomová práce, VUT FEKT v Brně, Brno, 2018, vedoucí práce prof. Ing. Vladislav Musil, CSc.

[115] Yu, S.-Y.; McCluskey, E. J.: On-line testing and recovery in TMR systems for real-time applications. *Proceedings International Test Conference 2001*, 2001: s. 240–249, ISSN 1089-3539, doi:10.1109/TEST.2001.966639.

[116] Yu, S. Y.; McCluskey, E. J.: Permanent fault repair for FPGAs with limited redundant area. *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, 2001: s. 125–133, ISSN 10636722, doi:10.1109/dftvs.2001.966761.

[117] Zhao, Z.; Agiakatsikas, D.; Nguyen, N. T.; aj.: Fine-grained module-based error recovery in FPGA-based TMR systems. *Proceedings of the 2016 International Conference on Field-Programmable Technology, FPT 2016*, ročník 11, č. 1, 2018: s. 101–108, doi:10.1109/FPT.2016.7929433.

Seznam obrázků

2.1 Posloupnost vzniku a šíření poruchy, chyby a selhání systému 10
2.2 Vanová křivka četnosti poruch elektronického systému [85] 11
2.3 Ukazatelé spolehlivosti MTTF, MTTR a MTBF 12
2.4 Radiační poruchy způsobené SEE jevy ovlivňující obvody FPGA 15
2.5 Vznik poruchy typu SEU v polovodičové součástce 16
2.6 Konceptuální rozložení vrstev obvodů FPGA 19
2.7 Struktura obvodu FPGA 19
2.8 Propojovací architektura FPGA 21
2.9 Struktura CLB 21
2.10 Architektura řezu (slice) v architektuře Xilinx Virtex-5 21
2.11 Schéma organizace konfigurační paměti FPGA 23
2.12 Návrhový proces s využitím nástrojů od firmy Xilinx 24
2.13 Proces syntézy, mapování, rozmístění a propojení logických prvků obvodu . 25
2.14 Rozdílová částečná rekonfigurace logiky v FPGA 26
2.15 Statický návrh 26
2.16 Modulární dynamický návrh 26
2.17 Poruchy způsobené vlivem SEU v obvodech SRAM FPGA 29
2.18 Taxonomie technik typických pro systémy odolné proti poruchám 30
2.19 Architektura duplexního systému s porovnáním výstupů 32
2.20 Porovnání spolehlivosti DMR a nezabezpečeného systému 32
2.21 Architektura systému TMR 33
2.22 Porovnání spolehlivosti TMR, NMR a nezabezpečeného systému 34
2.23 Metody pro zvýšení odolnosti proti poruchám používané v systémech implementovaných do FPGA v praxi 34
2.24 Taxonomie technik periodického čištění konfigurační paměti 35
2.25 Schéma časování periodického čištění konfigurační paměti 36
2.26 Schéma časování periodického čištění s vyčítáním konfigurační paměti . . . 36
2.27 Způsoby adresování konfigurační paměti během periodického čištění 37
2.28 Návrh externího a interního scrubbingu 38

3.1 Synchronizace stavu redundantních obvodů FPGA [83] 43
3.2 Architektura víceprocesorového systému MPSoC [74] 44
3.3 Časový diagram rekonfigurace a synchronizace ve více-procesorovém systému 44
3.4 Návrh synchronizace procesoru MicroBlaze se sdílenou pamětí BRAM [43] . 45
3.5 Časový diagram rekonfigurace a synchronizace procesoru MicroBlaze [43] . 46
3.6 Návrh synchronizace procesoru Plasma pomocí sdílené paměti BRAM [32] . 46
3.7 Návrh synchronizace procesoru PicoBlaze pomocí sběrnice Wishbone s přivedením výstupů do bezpečného stavu [6] 47

3.8 Návrh synchronizace procesoru ASIP pomocí vzájemně propojených redundantních pamětí BRAM a registrů se zpětnými vazbami 48
3.9 Synchronizace konečných automatů s předvídáním stavu [8] 49
3.10 Architektura pro obnovu stavu registrů na úrovni RTL pomocí volícího a přímo přepisujícího schématu [115] 50
3.11 Opravná logika volícího a přímo přepisujícího schématu pro obnovu stavu registrů [115] 50
3.12 Schéma opravné logiky ScTMR [24] 51
3.13 Schéma zabezpečeného a opravného návrhu pro obnovu stavu systému TMR na úrovni RTL [81] 52

5.1 Lokální TMR 58
5.2 Globální TMR 58
5.3 Distribuované TMR 58
5.4 Ztrojený majoritní volič 59
5.5 Redukující majoritní volič 59
5.6 Oddělující majoritní volič 60
5.7 Nesynchronizující volič 60
5.8 Synchronizující volič 60
5.9 Způsoby vkládání TMR v rámci návrhového procesu 62
5.10 Návrhový proces nástroje BL-TMR 63
5.11 Návrhový proces rekonfigurovatelného systému pro obvody od firmy Xilinx 66
5.12 Vytvoření rekonfigurovatelných regionů v konfigurační paměti FPGA 67
5.13 Příklad vytvoření PRR v jednom a nebo více hodinových regionech FPGA . 68
5.14 Jednotka pro řízení částečné dynamické rekonfigurace 71
5.15 Principy simulace poruch v konfigurační paměti FPGA 73
5.16 Provedení experimentů s injekcí poruch SEU 73

6.1 Výběr metody synchronizace stavu 76
6.2 Výběr vhodného synchronizačního stavu 78
6.3 Základní typy paměťových prvků v FPGA 79
6.4 Proces opravy stavu systému z poruchy 83

7.1 Model stavového automatu 85
7.2 Architektura řídicího systému sběrnice CAN 86
7.3 Zjednodušená schémata stavových automatů v řadiči sběrnice CAN 87
7.4 Schéma řadiče sběrnice CAN zabezpečeného pomocí TMR 88
7.5 Časový diagram opravy a synchronizace řadiče sběrnice CAN z poruchy . . 89
7.6 Ručně upravené regitry pro synchronizaci na úrovni RTL 91
7.7 Schéma synchronizace stavu pomocí sériového propojení 91
7.8 Schéma synchronizace stavu pomocí paralelní sběrnice 92

8.1 Model mikrokontroleru a aritmeticko-logické jednotky 95
8.2 Úrovně synchronizace provádění procesoru 97
8.3 Synchronizace procesoru se sdílenou datovou pamětí 98
8.4 Synchronizace procesoru se sdílenou synchronizační pamětí 98
8.5 Synchronizace procesoru pomocí sběrnice 99
8.6 Způsoby programové implementace synchronizace procesoru 101
8.7 Blokové schéma mikrokontroleru NEO430 102

8.8 Architektura jádra procesoru NEO430 103
8.9 Stavový diagram uvažovaného systému opravitelného z poruchy 104
8.10 Základní schéma modulu odolné architektury ReSyTMR 104
8.11 Logika komparátoru TMR 105
8.12 Jednotka FT Manager 105
8.13 Implementace stavového automatu jednotky FTM 106
8.14 Rozvržení rekonfigurovatelné architektury ReSyTMR v FPGA 107
8.15 Implementace jednotky arbitru synchronizace pro synchronizační restart . . 108
8.16 Schéma modulu odolné architektury ReSyTMR se sdílenou datovou pamětí 109
8.17 Schéma synchronizace redundantních procesorů pomocí přerušení 110
8.18 Schéma rozložení programu v pamětech mikrokontroleru NEO430 110
8.19 Simulace synchronizace registrů procesoru během obsluhy přerušení 112
8.20 Schéma experimentálního prostředí pro ověření zabezpečení procesoru . . . 113
8.21 Schéma architektury jednotky PDR Framework 114
8.22 Porovnání spolehlivosti nezabezpečeného systému, CG-TMR, CG-TMR s periodickým čištěním a ReSyTMR 117
8.23 Porovnání spolehlivosti CG-TMR s periodickým čištěním, ReSyTMR a FG-TMR 118

Seznam tabulek

2.1 Porovnání hlavních rysů technologií FPGA [3] [56] 17
2.2 Technologický vývoj obvodů FPGA od firmy Xilinx [3][4][109] 18
2.3 Porovnání parametrů CLB u řad FPGA Xilinx Virtex 20
2.4 Porovnání parametrů vybraných obvodů FPGA od firmy Xilinx 20
2.5 Porovnání parametrů konfiguračních rozhraní u rodiny FPGA Virtex [106] . 22
2.6 Adresa konfiguračního rámce FAR u obvodů FPGA Xilinx Virtex 23
2.7 Příklad hodnot výskytu poruch SEFI a SEU na orbitě u LEO a GEO [82] . 29

3.1 Vyhodnocení vhodnosti metod pro synchronizaci v TMR pro různé obvodové platformy 52

5.1 Porovnání hlavních rysů CG-TMR a FG-TMR 57
5.2 Přehled nástrojů pro automatické generování vkládání TMR 62
5.3 Porovnání metod pro opravu stavu systému pomocí rekonfigurace FPGA . . 70
5.4 Režie implementace jednotky PDR Unit a řadiče rekonfigurace GPDRC . . 72

7.1 Délka trvání operací prováděných v řadiči sběrnice CAN 93
7.2 Délka rekonfigurace a synchronizace stavu řadiče sběrnice CAN 93
7.3 Přehled spotřebovaných zdrojů FPGA implementací řadiče sběrnice CAN . 94

8.1 Porovnání implementace synchronizace stavu v procesoru 97
8.2 Logická funkce komparátoru TMRC 105
8.3 Přehled vybraných příkazů uživatelského terminálu jednotky PDR Framework 115
8.4 Velikost implementace mikrokontroleru s procesorem NEO430 v FPGA . . . 115
8.5 Velikost modulu ReSyTMR a jeho komponent v FPGA 116
8.6 Porovnání režie implementace systému při různých způsobech zabezpečení . 116
8.7 Porovnání přístupu k synchronizovaných objektům u různých procesorů (R – čtení, W – zápis) 119

Seznam použitých zkratek

ASIC	Application-Specific Integration Circuit – integrovaný obvod pro specifické použití
ASIP	Application-Specific Instruction set Processor – procesor s aplikačně specifickou instrukční sadou
BL-TMR	BYU-LANL Triple Modular Redundancy – nástroj pro automatickou implementaci FG-TMR vyvinutý univerzitou Brigham Young
CAN	Controller Area Network – průmyslová sběrnice CAN
CG-TMR	Coarse-Graine TMR – hrubozrnná architektura TMR
CLB	Configurable Logic Block – konfigurovatelný logický blok u FPGA
CPLD	Complex Programmable Logic Device – komplexní programovatelný logický obvod
CPU	Central Processing Unit – centrální řídící jednotka
DFT	Design For Testability – návrh pro snadnou testovatelnost
DMR	Dual Modular Redundancy – dvojitá modulární redundance
DSP	Digital Signal Processor – digitální signálový procesor
DTMR	Distributed TMR – distribuované TMR
DUT	Design Under Test – testovaný návrh systému
EASA	European Union Aviation Safety Agency – Evropská agentura pro bezpečnost letectví
ECC	Error Correction Code – korekční kód pro opravu chyb
ECSS	European Cooperation for Space Standardization – Evropská kooperace pro standardizaci ve vesmíru
EEPROM	Electrically Erasable PROM – elektricky vymazatelná paměť PROM
EPROM	Erasable PROM – vymazatelná paměť PROM
ESA	European Space Agency – Evropská vesmírná agentura
FAA	Federal Aviation Administration – Federální letecká správa

FG-TMR	Fine-Graine TMR – jemnozrnná architektura TMR
FPGA	Field Programmable Gate Array – programovatelné hradlové pole
FPLA	Field Programmable Logic Array – programovatelné logické pole
FSM	Finite State Machine – stavový automat
GAL	Generic Array Logic – obecné pole logiky
GCR	Galactic Cosmic Ray – galaktické kosmické záření
GPDRC	Generic Partial Dynamic Reconfiguration Controller – generický řadič rekonfigurace
GPDRC	Generic Partial Dynamic Reconfiguration Controller – generický řadič rekonfigurace
GPU	Graphics Processing Unit – grafický procesor
GTMR	Global TMR – globální TMR
ICAP	Internal Configuration Access Port – interní konfigurační port obvodu FPGA
IRQ	Interrupt Request – požadavek na přerušení procesoru
LSI	Large Scale Integration – vysoký stupeň integrace
LTMR	Local TMR – lokální TMR
LUT	Lookup Table – vyhledávací tabulka v CLB realizující logickou funkci
MBU	Multiple Bit Upset – efekt způsobený působením kosmického záření, který představuje náhodnou změnu hodnoty několika bitu v paměťových prvcích obvodu
MCU	Microcontroller Unit – mikrokontroler
MCU	Multiple Cell Upset – náhodná několikanásobná změna paměťových buněk způsobená poruchou
MTBF	Mean Time Between Failures – střední doba mezi poruchami
MTTF	Mean Time To Failure – střední doba do poruchy
MTTR	Mean Time To Repair – střední doba do opravy
NGC	Native Generic Compiler – soubor s popisem číslicového návrhu, který je výstupem logické syntézy
NGD	Native Generic Database – soubor s popisem číslicového návrhu, který je výstupem procesu mapování
PAL	Programmable Array Logic – programovatelné pole logiky
PDR	Partial Dynamic Reconfiguration – částečná dynamická rekonfigurace obvodů FPGA

PLA	Programmable Logic Array - programovatelné logické pole
PLD	Programmable Logic Device – programovatelný logický obvod
PRM	Partial Reconfiguration Module – částečně rekonfigurovatelný modul
PROM	Programmable Read Only Memory – programovatelná paměť ROM
PRR	Partial Reconfiguration Region – částečně rekonfigurovatelný region
RAM	Random Access Memory – paměť s náhodným přístupem
ReSyTMR	Reconfigurable Synchronizable TMR – architektura TMR podporující rekonfiguraci a synchronizaci stavu
ROM	Read Only Memory – paměť pouze pro čtení
RTL	Register transfer level – úroveň meziregistrových přenosů
SCR	Sun Cosmic Ray – sluneční kosmické záření
SEE	Single Event Effects – skupina efektů, které způsobují poruchy v převážně polovodičových součástkách způsobené působením kosmického záření
SEFI	Single Event Functional Interrupt – efekt způsobený působením kosmického záření, který nejčastější v podobě poruchy typu SEU způsobuje změnu funkce číslicového systému nebo jeho části
SET	Single Event Transient – efekt způsobený působením kosmického záření, který způsobuje nežádoucí zákmity v kombinační logice obvodu
SEU	Single Event Upset – efekt způsobený působením kosmického záření, který představuje náhodnou změnu hodnoty bitu v paměťových prvcích obvodu
SoC	System on Chip – systém integrovaný na jednom čipu
SPI	Serial Peripheral Interface – sériové periferní rozhraní
SPLD	Simple Programmable Logic Device – jednoduchý programovatelný logický obvod
SRAM	Static Random Access Memory – statická paměť s přímým přístupem nebo paměť s libovolným výběrem
TMR	Triple Modular Redundancy – trojitá modulární redundance
UCF	User Constraints File – soubor omezení pro implementaci systému do FPGA
VHDL	VHSIC Hardware Description Language - VHSIC jazyk pro popis hardware
VLSI	Very Large Scale Integration – velmi vysoký stupeň integrace