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

AUTOREFERÁT DISERTAČNÍ PRÁCE
PHD THESIS AUTOREFERATE

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.

Klíčová slova

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

Citace

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

Obsah

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. 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. 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.

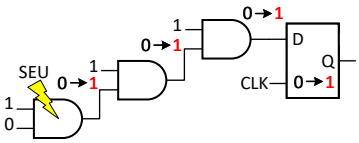
Programovatelné logické obvody FPGA (Field Programmable Gate Array) se postupně staly oblíbenou obvodovou platformou pro různorodé aplikace ve všech průmyslových odvětví trhu. Výhodou moderních obvodů SRAM FPGA je možnost částečné dynamické rekonfigurace PDR (Partial Dynamic Reconfiguration), která umožňuje 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. PDR lze také použít v systémech odolných proti poruchám pro opravu poruch v konfigurační paměti FPGA.

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 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 hardwarových jednotek po vzniku poruchy.

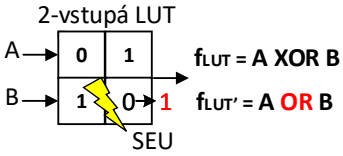
1.1 Poruchy číslicových systémů a FPGA

Současné číslicové systémy a programovatelné obvody FPGA obsahují milióny tranzistorů CMOS. Vzhledem k neustálému zvyšování počtu tranzistorů na čipu, zmenšování jeho plochy a zvyšování frekvence se některé integrované obvody a systémy na čipu staly mnohem náchylnější k poruchám, vznikajícím vlivem působení nepříznivého okolního prostředí a degradace elektronického materiálu [17], než dříve. Poruchy číslicových systémů jsou většinou způsobeny chybami nebo nedokonalostmi výrobního procesu, chybami a poškozením použitého polovodičového materiálu, dlouhodobým opotřebením součástek a působením vlivu okolního prostředí. Vzniklé poruchy mohou být trvalé nebo dočasné. 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. Působení energetických částic kosmického záření může ovlivnit funkčnost součástek a způsobit jejich poruchu. Mezi nejčastější přechodné poruchy, které mohou ohrozit spolehlivost číslicového systému v FPGA patří následující dvě poruchy:

- *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. Vliv poruchy typu SEU na hradla a bloky LUT v obvodech FPGA je znázorněn na obrázcích 1.1 a 1.2.
- *Single Event Transient (SET)* je přechodné napěťové nebo proudové rušení ovlivňující kombinační hradla. SET se může propagovat skrz hradla obvodu a eventuálně způsobit překlopení logického stavu paměťové buňky (tedy poruchu typu SEU).



Obrázek 1.1: SEU v log. hradlech



Obrázek 1.2: SEU v LUT

1.2 Systémy odolné proti poruchám

Redundance je klíčovým prvkem pro návrh systémů odolných proti poruchám. 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.

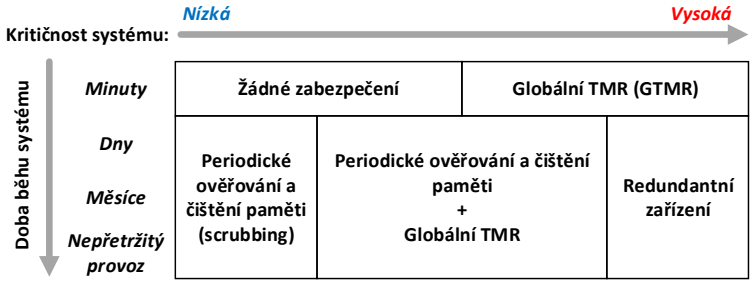
Trojitá modulární redundance (TMR) je nejpoužívanější technikou, jakou mohou být číslicové systémy zabezpečeny proti poruchám. Způsoby implementace TMR v číslicovém návrhu mohou být rozděleny dle granularity s jakou je architektura TMR využita pro zabezpečení systému. Podle množství zabezpečené logiky v FPGA a zrnitosti vložení TMR lze rozlišovat mezi hrubozrnnou a jemnozrnnou strukturou architektury TMR. Hrubozrnné TMR (CG-TMR) označuje implementaci TMR na úrovni celého systému nebo funkčního modulu. Naopak jemnozrnné TMR (FG-TMR) označuje implementaci TMR na úrovni zabezpečení jednotlivých logických obvodů nebo registrů. Implementace zabezpečení systému pomocí jemnozrnné architektury TMR se vyznačuje tím, že je snahou aplikovat techniku TMR na nejnižší logické úrovni návrhu. V tabulce 1.1 jsou porovnány parametry použití hrubozrnné a jemnozrnné architektury TMR.

Vlastnost	CG-TMR	FG-TMR
Maskování vícenásobných poruch	Ne	Ano
Schopnost lokalizace poruch	Malá	Velká
Vliv na max. hodinovou frekvenci	Malý	Velký
Režie způsobená redundancí	Velká	Velká
Automatizovaná implementace	Spíše ano	Ano
Verifikace správné implementace	Jednoduchá	Složitá

Tabulka 1.1: Porovnání hlavních rysů CG-TMR a FG-TMR

Nevýhoda CG-TMR se projeví při postupné akumulaci poruch typu SEU nebo výskytu několika poruch naráz. Pravděpodobnost poruchy jednoho modulu CG-TMR roste s velikosti zabezpečené logiky [3]. Majoritní volič umožňuje maskovat poruchu jednoho modulu CG-TMR. V případě výskytu poruchy v dalším modulu TMR selhává. Z tohoto důvodu se použití TMR kombinuje s technikami pro opravu poruch v FPGA.

Návrh systémů odolných proti poruchám do obvodů FPGA se v praxi může řídit podle doporučení od firmy Xilinx [1], znázorněného na obrázku 1.3. 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.



Obrázek 1.3: Metody pro zvýšení odolnosti proti poruchám používané v systémech implementovaných do FPGA v praxi

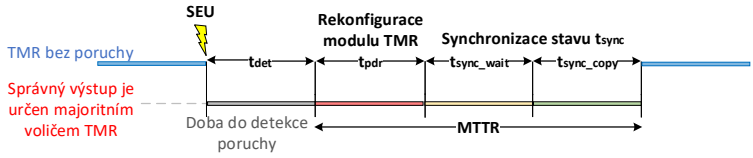
Pro opravu poruch v konfigurační paměti FPGA existují tři základní metody založené na schopnosti částečné dynamické rekonfigurace:

- Periodické ověřování a čištění konfigurační paměti (scrubbing) – Periodické čištění je spolu s implementací globálního TMR nejvhodnější technikou pro většinu případů použití.
- Rekonfigurace redundantních modulů – Alternativní metodou je využití částečné dynamické rekonfigurace pro opravu poruch v dedikovaných rekonfigurovatelných modulech systému zabezpečeného pomocí architektury TMR. Tento způsob rekonfigurace je schopen opravit poruchu ihned po její detekci.
- Relokace částečných bitstreamů pro opravu trvalé poruchy – Při opravě stavu systému v případě vzniku permanentní poruchy v konfigurační pamětí SRAM FPGA je částečná dynamická rekonfigurace využita pro přemístění rekonfigurovatelného modulu do jiné vhodné části konfigurační paměti FPGA, která není ovlivněna permanentní poruchou.

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. 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. Stav systému může být obnoven zpětně na základě předem uloženého obrazu správného stavu systému. 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 systému zabezpečeného pomocí TMR lze pokládat všechny metody založené na dopředné obnově stavu za metody provádějící synchronizaci stavu opravovaného modulu TMR s ostatními redundantními moduly.

Proces opravy stavu systému z poruchy pomocí rekonfigurace a synchronizace je znázorněn na obrázku 1.4. Doba trvání procesu opravy stavu systému je dána dle vzorce $t_{recov} = t_{det} + t_{pdr} + t_{sync}$ součtem časů potřebných pro detekci poruchy, opravu poruchy a synchronizaci stavu systému.

- *Doba detekce poruchy* t_{det} – Doba detekce poruchy je proměnná a závisí na tom, jak rychle je systém schopen poruchu detekovat.
- *Doba rekonfigurace* t_{pdr} – Doba rekonfigurace modulu PRM.
- *Doba synchronizace* t_{sync} – Výpočet doby synchronizace může být proveden pomocí vzorce $t_{sync} = t_{sync\_wait} + t_{sync\_copy}$. 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.



Obrázek 1.4: Proces opravy stavu systému z poruchy

2. Cíle disertační práce

Cílem této disertační práce je *vytvoření metodiky pro návrh a implementaci číslicových obvodů pro synchronizaci stavu rekonfigurovatelných modulů TMR architektury systému odolného proti poruchám implementovaného do FPGA*. Předpokladem pro řešení disertační práce bylo 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.

3. Návrh opravovaného systému odolného proti poruchám

Tato disertační práce se zaměřuje na opravu stavu systému s využitím rekonfigurace redundantních modulů 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 úspěšně rekonfigurován. V tabulce 3.1 je porovnána technika rekonfigurace CG-TMR s technikou periodického čištění a FG-TMR.

Vlastnost	Rekonfigurace 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 3.1: Porovnání metod opravy stavu systému rekonfigurací FPGA

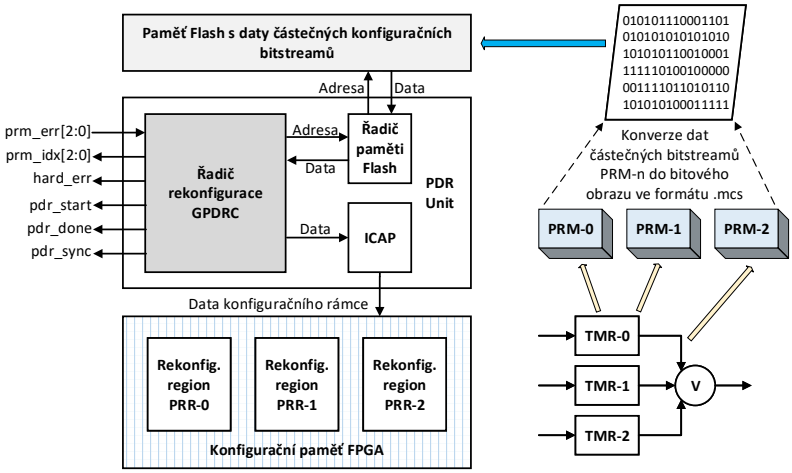
Nevýhodou periodického čištění může být právě periodické provádění, protože oprava poruchy může být provedena až v další periodě čištění konfigurační paměti. Naopak při použití rekonfigurace modulů TMR může být porucha opravena ihned, jakmile je detekována. Nicméně tato technika neumožňuje detekci latentních (skrytých) poruch, jež mohou být naopak detekovány a automaticky opraveny 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.

3.1 Řadič rekonfigurace

Řadič rekonfigurace je funkční jednotka zodpovědná za řízení částečné dynamické rekonfigurace FPGA. Tato disertační práce využívá řadič rekonfigurace *GPDRC (Generic Partial Dynamic Reconfiguration Controller)*,

který byl vyvinut v rámci aktivit [9] [5] naši výzkumné skupiny. 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. Pro rekonfiguraci je potřebné vygenerovat částečné konfigurační bitstreamy pro rekonfigurovatelné moduly 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 [9] [5]. 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 3.1.



Obrázek 3.1: 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, 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 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*.

3.2 Simulace poruch v konfigurační paměti

Při experimentech provedených v rámci této disertační práce byl pro simulaci poruch typu SEU v konfigurační paměti FPGA využit SEU framework, který byl vyvinut v rámci aktivit naši výzkumné skupiny [10]. 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 [4]. Cílem analýzy konfiguračních bitů číslicového návrhu je nalezení tzv. kritických bitů. Tyto bity ovlivňují přímo funkci implementované logiky uvnitř konfiguračních bloků FPGA.

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. Cyklus injekce poruch je ukončen v momentě ztráty komunikace se systémem.

4. Metodika synchronizace rekonfigurovatelných modulů

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 také na již existující návrh rekonfigurovatelné architektury TMR, nicméně návrh systému musí být pro umožnění synchronizace stavu mezi rekonfigurovatelnými moduly upraven.

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.

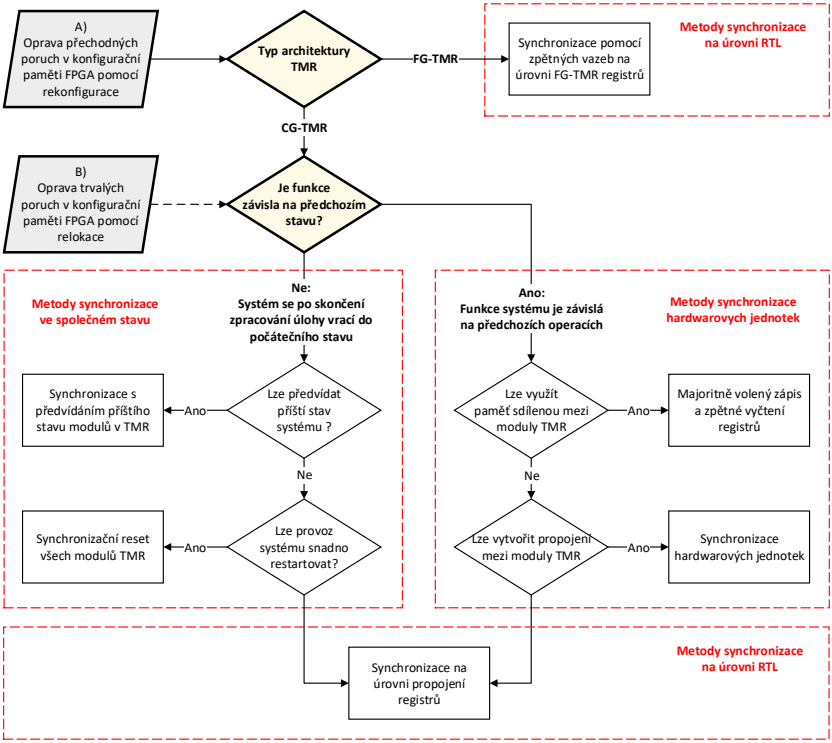
Na základě vhodnosti různých způsobů synchronizace pro různé typy systémů, lze rozdělit metody synchronizace do tří hlavních kategorií:

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.

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 4.1.



Obrázek 4.1: 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.

4.1 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 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.

Návrh a implementace vybrané metody synchronizace jsou úzce spjaty s architekturou cílového systému a jeho funkcí. Způsob návrhu metody synchronizace v rámci popsaných problémů i samotná implementace logiky provádějící synchronizaci mají výrazný vliv na funkci systému a jeho parametry. Přehled ovlivněných parametrů systém je uveden v tabulce 4.1

Dynamické parametry	Statické parametry
• Vliv na funkci systému • Čas potřebný pro provedení synchronizace	• Obvodová režie implementace • Příkon • Redukce provozní frekvence • Spolehlivost obvodů

Tabulka 4.1: Parametry metod synchronizace stavu

4.1.1 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.

- *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.

- *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.

4.1.2 Krok 2: Definice synchronizovaných objektů

Stav systému je reprezentován hodnotami všech paměťových elementů hardwarových jednotek uvnitř systému (registry, klopné obvody typu D, vestavěné paměti). 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 není nutné synchronizovat. Naopak, paměťové elementy důležité pro funkci systému nebo obsahující aplikační data je nezbytné synchronizovat.

4.1.3 Krok 3: Návrh propojení modulů TMR

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 (společné nulování modulů, přenesení provozního stavu).

b) Propojení na úrovni registrů (paralelní propojení, sériové propojení).

c) Propojení na úrovni modulů (propojení redundantních systémů přes sdílenou paměť, propojení pomocí sběrnice).

5. Synchronizace stavu systému s dávkovým zpracováním

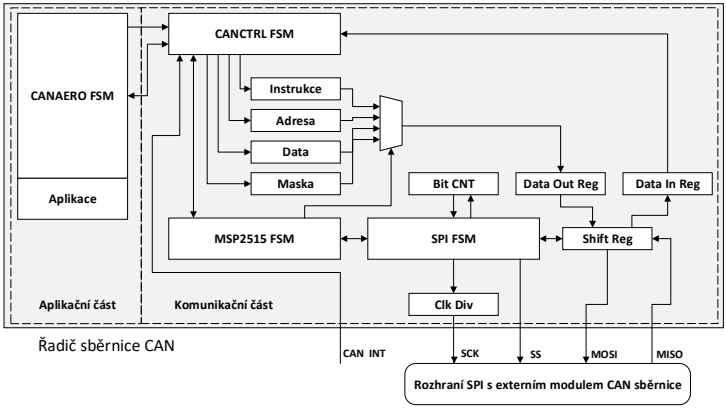
Systémy s dávkovým zpracováním dat jsou typicky aktivní pouze při zpracování vstupního bloku dat. Příkladem mohou být systémy provádějící periodické zpracování datových paketů síťové komunikace nebo obrazových informací. Tyto systémy se po dokončení své činnosti vrací do stavu nečinnosti, ve kterém čekají na příchod nových vstupních dat. Pro implementaci těchto systémů 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 [11] [12] [15] [16].

5.1 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 [11] [12]. Sběrnice CAN je sériová datová sběrnice, jež se stala široce rozšířeným průmyslovým standardem [7]. 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* [8]. 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.

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* 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 [11]. Architektura systému řadiče sběrnice CAN je znázorněna na obrázku 5.1.

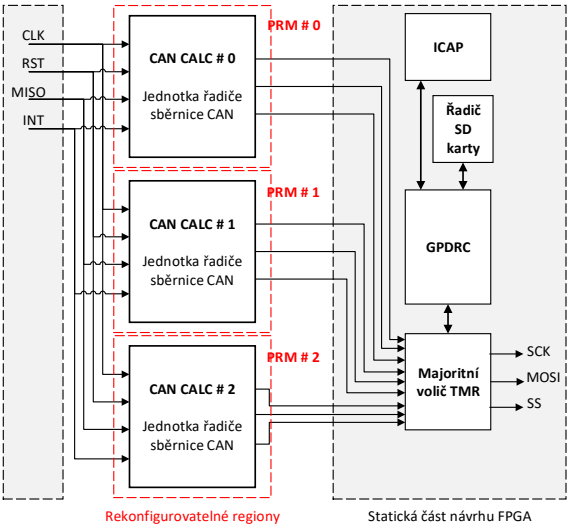


Obrázek 5.1: Architektura řídicího systému sběrnice CAN

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 5.1 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. 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 [8], 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. Aplikace umožňuje provádět distribuované matematické operace za pomoci služeb implementovaných za pomocí protokolu CANAerospace.

Řídicí systém byl nejdříve zabezpečen pomocí architektury CG-TMR s cílem zvýšení celkové spolehlivosti. Jednoduchý volič byl připojen na výstupy ztrojených instancí řadiče pro maskování poruch při komunikaci po rozhraní SPI s externím modulem. Následně byla architektura upravena pro podporu rekonfigurace řadičem GPDRC za běhu systému. Rekonfigurovatelná architektura celého systému je znázorněna na obrázku 5.2.

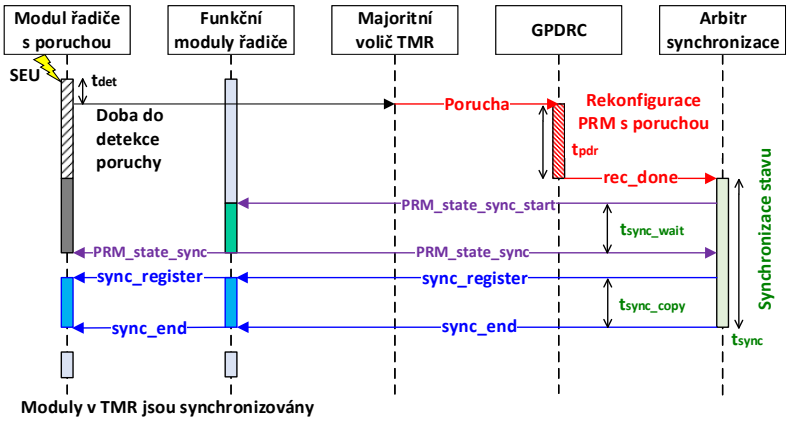


Obrázek 5.2: Rekonfigurovatelná architektura řadiče sběrnice CAN

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 5.3. 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.



Obrázek 5.3: Časový diagram opravy a synchronizace řadiče sběrnice CAN z poruchy

5.2 Návrh techniky synchronizace stavu

Návrh logických obvodů realizujících synchronizaci stavu v rekonfigurovatelné architektuře řadiče sběrnice CAN byl založen na principech popsaných v kapitole 4.1. Pro řízení synchronizace byly navrženy jednotky arbitra a řadiče synchronizace. Arbitr 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.

Pro návrh metody synchronizace stavu bylo v první řadě nutné analyzovat celý řídicí systém. 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.

5.2.1 Synchronizace stavu komunikační vrstvy řadiče

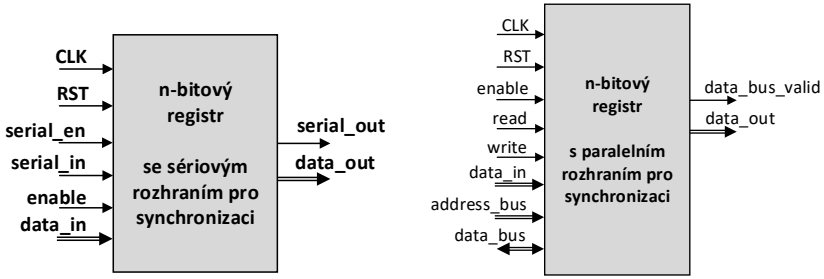
Princip synchronizace sekvenčních automatů v komunikační vrstvě řadiče je založen na čekání na okamžik, kdy sekvenční automaty v referenčním systému přejdou do specifického stavu, a nastavení stavu synchronizovaného systému do téhož stavu.

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.

5.2.2 Synchronizace stavu aplikační vrstvy řadiče

Jednotka CANAERO disponuje několika datovými registry, které uchovávají stav aplikace. 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 znázorněna na obrázcích 5.4a a 5.4b.



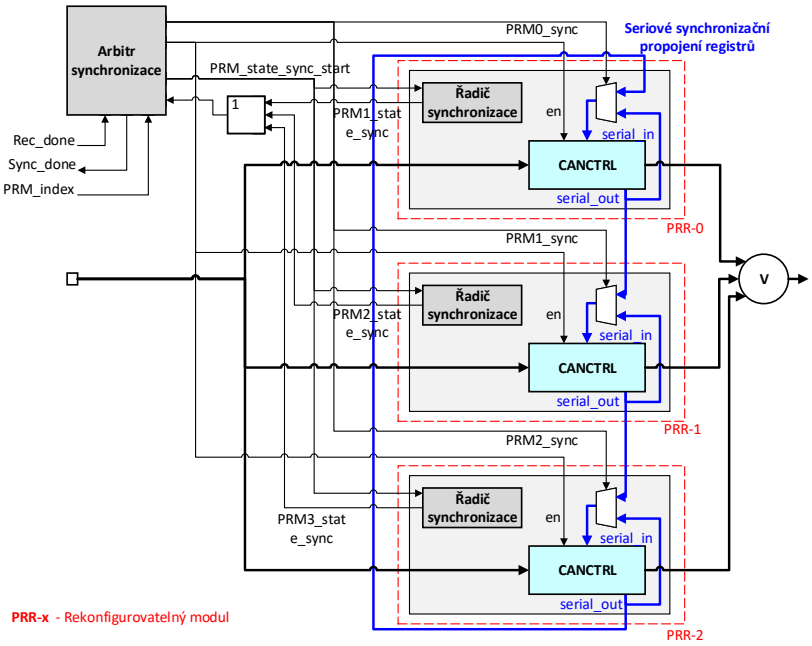
(a) Registr pro sériovou synchronizaci (b) Registr pro paralelní synchronizaci

Obrázek 5.4: Ručně upravené regitry pro synchronizaci na úrovni RTL

Synchronizace 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 5.4a. 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 v synchronizovaném modulu. Při synchronizaci je výstup *serial_out* bezporuchových modulů přiveden zpět na vstup *serial_in* stejného modulu.

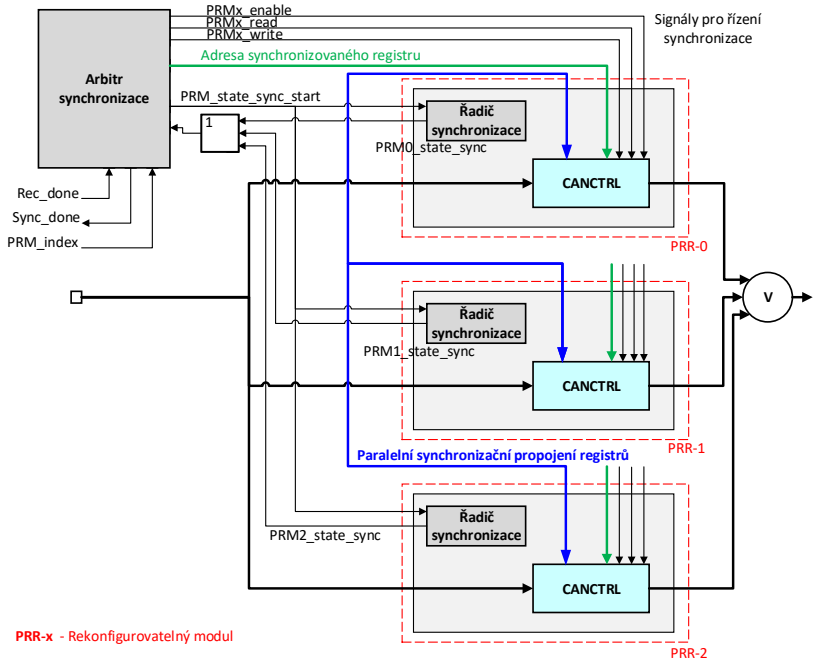


Obrázek 5.5: Schéma synchronizace stavu pomocí sériového propojení

Synchronizace 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 5.4b. 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 5.6. 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 5.6: Schéma synchronizace stavu pomocí paralelní sběrnice

5.3 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 5.1 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.

Varianty systému	Sériové propojení		Paralelní propojení	
Virtex5 XC5VSX50T 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 5.1: Přehled spotřebovaných zdrojů FPGA implementací řadiče sběrnice CAN

V tabulce 5.2 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.

V tabulce 5.3 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
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 5.2: Délka trvání operací prováděných v řadiči sběrnice CAN

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 5.3: 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.

6. Synchronizace operačního stavu procesorového systému

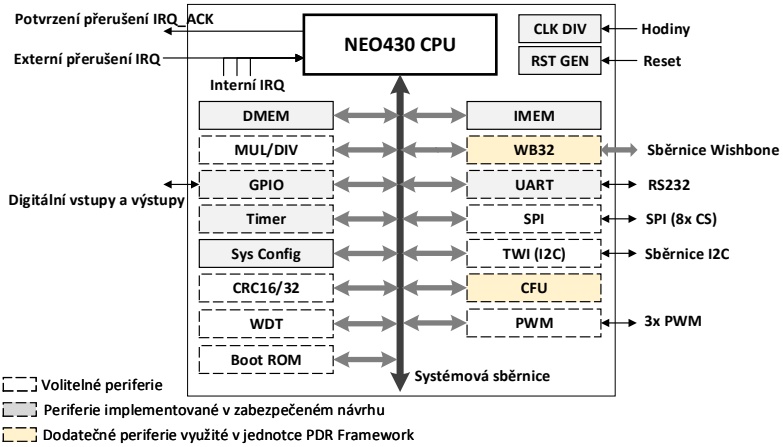
Tato kapitola demonstruje použití metodiky pro návrh synchronizace stavu na systému řízeném jádrem procesoru NEO430, který je zabezpečen pomocí rekonfigurovatelné architektury CG-TMR.

Při návrhu metody synchronizace stavu pro procesor je nutné uvažovat nad synchronizací vnitřních registrů, které jsou součástí implementované architektury procesoru a nad synchronizací registrů a paměťových objektů, které definují stav procesoru z hlediska programátora. Z tohoto pohledu je u každého procesoru nutné synchronizovat následující objekty: programový čítač PC (Program Counter), ukazatel na zásobník SP (Stack Pointer), řídicí a stavový registr, všeobecné registry, programový zásobník a data uložená v operační paměti.

6.1 Mikrokontrolér NEO430

Mikrokontroler NEO430 je syntetizovatelná komponenta obsahující jádro 16-bitového procesoru NEO430, který je kompatibilní s procesorem TI MSP430 [6]. 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 6.1. 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. Řídící automat procesoru po restartu přechází do stavu *IFETCH0*, ve kterém zpracování všech instrukcí začíná. Tento stav slouží také jako výchozí stav v případě, že procesor pracuje v klidovém režimu[1].

[1] Klidový režim byl využit pro synchronizaci provádění programu procesory.

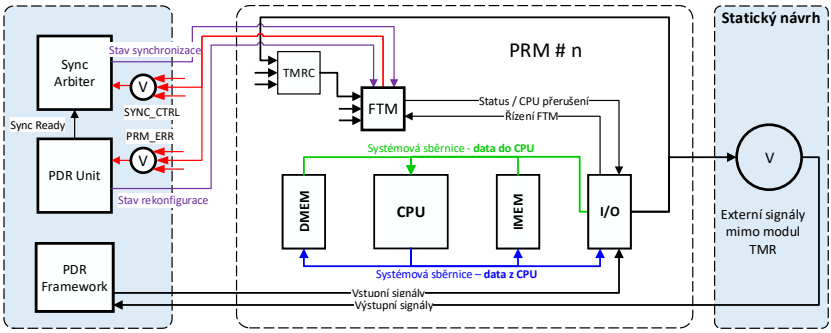


Obrázek 6.1: Blokové schéma mikrokontroleru NEO430

6.2 Návrh 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. Schéma jednoho modulu architektury ReSyTMR je znázorněno na obrázku 6.2. 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.

Monitor stavu odolného systému *FTM (Fault Tolerant Monitor)* byl navržen pro řízení stavu systému odolného proti poruchám v rámci architektury ReSyTMR. Vstup FTM je propojen s výstupem TMRC, který porovnává výstupy všech redundantních modulů a případnou poruchu



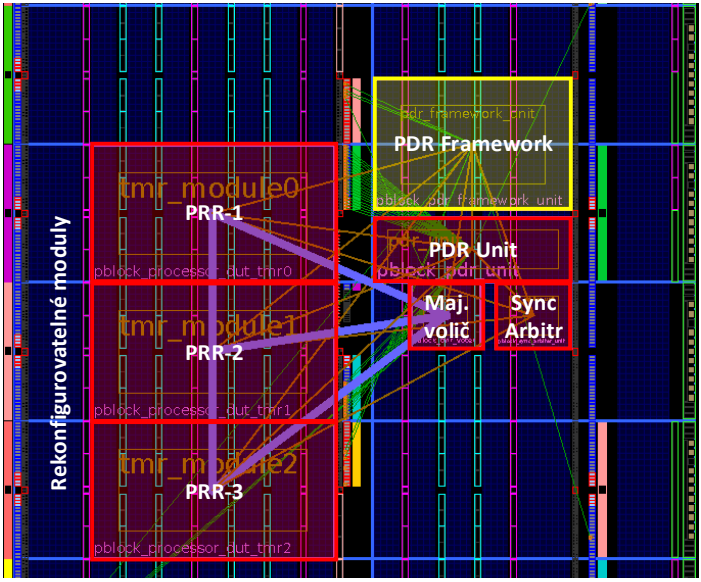
Obrázek 6.2: Základní schéma modulu odolné architektury ReSyTMR

indikuje chybovým vektorem. Současně je FTM schopný zareagovat na selhání rekonfigurace nebo synchronizace. FTM o detekované poruše informuje připojený procesor. Nicméně proces opravy řídí FTM autonomně.

Rozvržení rekonfigurovatelné architektury ReSyTMR v konfigurační paměti FPGA je znázorněno na obrázku 6.3. 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. Rekonfigurovatelná oblast je rozdělena do tří rekonfigurovatelných regionů PRR-1, PRR-2 a PRR-3. Každý rekonfigurovatelný region obsahuje jednu instanci modulu architektury ReSyTMR.

6.3 Implementace metod pro synchronizaci

Na základě vyhodnocení vhodnosti možných řešení synchronizace stavu (tabulka 6.1) pro mikrokontroler NEO430 zabezpečený pomocí architektury ReSyTMR byly navrženy a implementovány dvě metody synchronizace stavu procesoru využívající synchronizační restart a sdílenou paměť. Návrh synchronizace s využitím propojení na úrovni registrů (paralelní nebo sériové) by pro procesor znamenal velký zásah do jeho architektury a způsobil by značnou implementační režii. Pokud se zaměříme na metody synchronizace stavu procesoru provádějící kopii a přenos synchronizačních objektů z bezporuchového procesoru do procesoru, který byl rekonfiguro-



Obrázek 6.3: Rozvržení rekonfigurovatelné architektury ReSyTMR v FPGA

ván, pak je nutno při návrhu techniky synchronizace jednotlivých typů synchronizačních objektů zvážit způsob propojení procesorů v redundantních modulech TMR. Pro synchronizaci procesoru je nejvhodnější použít propojení přes sdílenou paměť.

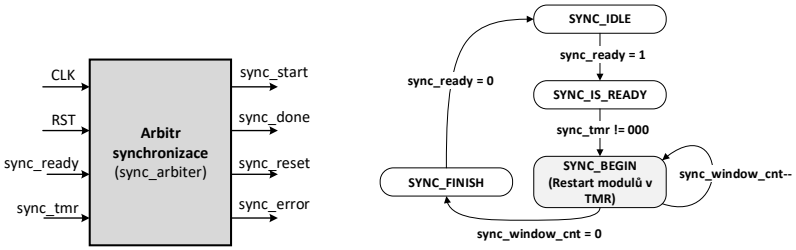
Metoda synchronizace	Úprava procesoru	Režie HW	Režie SW
Synchronizační restart	Ne	Malá	Malá
Přenesení stavu	Není vhodné	–	–
Paralelní propojení	Velká	Velká	Žádná
Sériové propojení	Velká	Velká	Žádná
Sdílená paměť	Malá	Malá	Větší
Synchronizační sběrnice	Ne	Malá	Velká

Tabulka 6.1: Porovnání implementace synchronizace stavu v procesoru

6.3.1 Synchronizace pomocí synchronizačního restartu

Pro implementaci metody synchronizace pomocí synchronizačního restartu nebylo nutné provádět žádné změny v základní architektuře odolné architektury ReSyTMR prezentované na obrázku 6.2. Blokové schéma implementované jednotky arbitra synchronizace je znázorněno na obrázku 6.4a.

Jednotka arbitru pracuje podle stavového automatu uvedeného na obrázku 6.4b. 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.

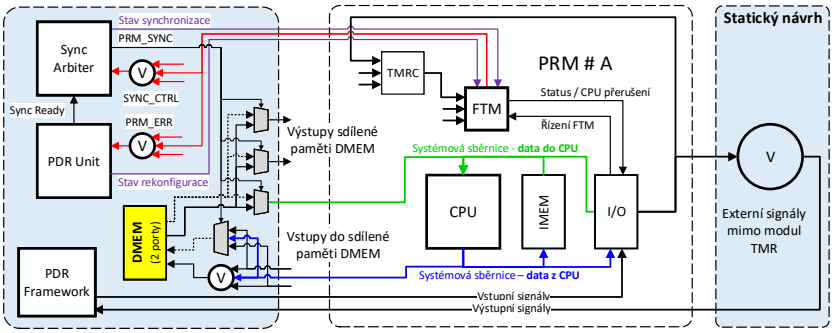


(a) Blokové schéma (b) Stavový automat

Obrázek 6.4: Implementace jednotky arbitru synchronizace pro synchronizační restart

6.3.2 Synchronizace pomocí sdílené datové paměti

Schéma architektury ReSyTMR s upraveným návrhem mikrokontroleru se sdílenou datovou pamětí je znázorněno na obrázku 6.5. 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.

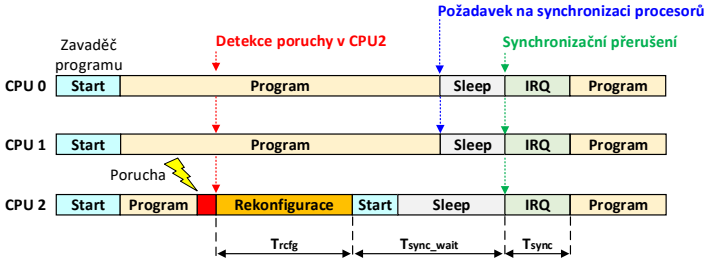


Obrázek 6.5: Schéma modulu odolné architektury ReSyTMR se sdílenou datovou pamětí

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. Tento problém byl vyřešen implementací druhého přístupového portu ke sdílené paměti DMEM, který umožňuje synchronizovanému procesoru pouze čtení dat.

Způsob synchronizace běhu procesorů pomocí synchronizačního přerušení je znázorněn na diagramu na obrázku 6.6. 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 dokončit aktuálně rozpracované instrukce.

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. 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. Pro přechod do klidového režimu je nutná aktivace digitálního vstupu, na který je přiveden signál z arbitru synchronizace.



Obrázek 6.6: Schéma synchronizace procesorů pomocí přerušení

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 synchronizačního 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.

6.4 Experimentální a implementační výsledky

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*. PDR Framework je taktéž řízen mikrokontrolerem s jádrem procesoru NEO430, který je rozšířen o druhé sériové rozhraní UART a

sadu záchytných registrů a čítačů umožňujících zachytit vstupní signály připojené z verifikovaného číslicového návrhu.

Implementační režie jednoho modulu architektury ReSyTMR a jeho komponent je uvedena v tabulce 6.3. Schéma modulu a jeho propojení se statickou logikou v rámci rekonfigurovatelného návrhu je znázorněno na obrázku 6.2 v kapitole 6.1.

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

Tabulka 6.2: 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 6.3: 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 6.4. 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.

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 6.4: Porovnání režie implementace systému při různých způsobech zabezpečení

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 do konfigurační paměti FPGA s následujícími variantami různě zabezpečeného systému:

a) *Simplex* – nezabezpečený systém mikrokontroléru,

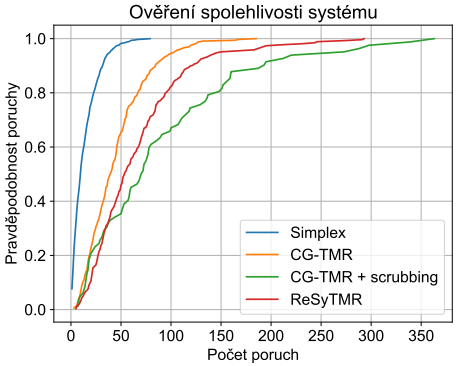
b) *CG-TMR* – systém zabezpečený pomocí CG-TMR (implementace mikrokontroléru byla v číslicovém návrhu ztrojena, majoritní volič byl implementován mimo oblast injektovaných poruch v konfigurační paměti FPGA),

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

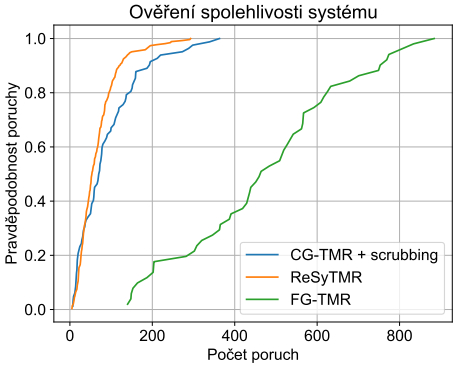
d) *CG-TMR + scrubbing* – systém zabezpečený pomocí CG-TMR s periodickým čištěním konfigurační paměti. Periodické čištění bylo simulováno pomocí programu v PC, který po každém cyklu 20 poruch přeprogramoval oblast s návrhem CG-TMR původním konfigurací.

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 6.7. Porovnání spolehlivosti implementace návrhu zabezpečeného pomocí architektury ReSyTMR a návrhu zabezpečeného pomocí FG-TMR je znázorněno na obrázku 6.8.



Obrázek 6.7: Porovnání spolehlivosti nezabezpečeného systému, CG-TMR, CG-TMR s periodickým čištěním a ReSyTMR



Obrázek 6.8: 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 6.4 dochází až k 50% poklesu maximální pracovní frekvence.

7. Závěr

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.

Navržená metodika popisuje všechny kroky návrhu metody synchronizace stavu vzhledem k vlastnostem a architektuře cílového systému. 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 systému. V práci byly 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 mikrokontrolérem 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 [12] [15] [16], 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 5. Článek [16] definoval základní opěrné body navržené metodiky, blíže prezentované v kapitole 4. 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 6. Výsledky výzkumu byly publikovány ve dvou vědeckých článcích [13] a [14] na mezinárodních konferencích. Navržená metodika byla díky znalostem získaných při návrhu a implementaci zabezpečení pro experimentální systém řízený mikrokontrolérem NEO430 dále rozšířena do finální podoby.

7.1 Přínos práce

Hlavním přínosem disertační práce je vytvoření metodiky pro návrh synchronizace stavu rekonfigurovatelných modulů. Navržená metodika spolu s výsledky výzkumné práce 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 FPGA a mechanismy opravy stavu systému z poruchy.
- Byla popsána problematika implementace TMR na různých úrovních návrhu. Součástí experimentů bylo vytvoření návrhu mikrokontroléru zabezpečeného pomocí architektury FG-TMR na úrovni RTL, která byla automaticky implementována nástrojem BL-TMR.
- Byla vytvořena metodika pro návrh a implementaci číslicových obvodů pro synchronizaci stavu rekonfigurovatelných modulů architektury CG-TMR systému odolného proti poruchám implementovaného do FPGA. Navržená metodika doplňuje řešení problematiky návrhu rekonfigurovatelného systému zabezpečeného pomocí architektury CG-TMR.
- 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).

- Pro ověření správné funkce navržených mechanismů opravy stavu systému bylo implementováno verifikační prostředí PDR Framework. Jednotka PDR Framework tvoří prostředníka umožňující komunikaci mezi experimentálním PC (návrhářem), zabezpečeným systémem a číslicovými obvody zodpovědnými za opravu stavu systému. PDR Framework implementuje rozhraní umožňující provádět různé příkazy pro řízení a monitorování experimentů.

- 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.

7.2 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.

Synchronizace stavu odolného systému může být potřebná také v případě opravy trvalé poruchy s využitím techniky relokace konfiguračních bitstreamů, kdy je v rámci rekonfigurovatelné architektury TMR nutné přesunout logiku jednoho redundantního modulu na jiné místo v konfigurační paměti FPGA. Výsledkem 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.

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 [2], který pracuje správně. Tato technika synchronizace stavu vyžaduje detailní znalost struktury konfigurační paměti a formátu konfiguračního bitstreamu pro daný typ obvodu FPGA.

Literatura

[1] 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).

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

[3] 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.

[4] 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.

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

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

Publikace autora

1. 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)*. Cluj-Napoca: IEEE Computer Society, 2019, s. 136-140. ISBN 978-1-7281-0073-9. (90%)

2. 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)*. Santiago: IEEE Computer Society, 2019, s. 32-35. ISBN 978-1-7281-1756-0. (90%)

3. Szurman, K.; Kotásek, Z.: Fault Recovery for Coarse-Grained TMR Soft-Core Processor Using Partial Reconfiguration and State Synchronization. In: *Proceedings of the 7th Prague Embedded Systems Workshop*. Roztoky u Prahy: Fakulta informačních technologií ČVUT, 2019, s. 6-7. ISBN 978-80-01-06607-2. (90%)

4. Szurman, K; Kotásek, Z.: State Synchronization of Faulty Soft Core Processors in Reconfigurable TMR Architecture. In: *Počítačové architektúry & diagnostika 2017*. Smolenice: Slovenská technická univerzita v Bratislavě, 2017, s. 51-54. ISBN 978-80-972784-0-3. (80%)

5. Szurman, K.; Mičulka, L.; Kotásek, Z.: Towards a State Synchronization Methodology for Recovery Process after Partial Reconfiguration of Fault Tolerant Systems. In: *Proceedings of the 4th Prague Embedded Systems Workshop*. Roztoky u Prahy, 2016. ISBN 978-80-01-05984-5. (70%)

6. 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*, IEEE Computer Society, Káhira, EG, 2014, s. 231-236, ISBN 978-1-4799-6594-6. Publikace získala ocenění **„Best Paper Award“**. (70%)

7. 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*, IEEE Computer Society, Verona, IT, 2014, s. 704-707, ISBN 978-1-4799-5793-4. (60%)

8. Szurman, K.: Synchronization Methodology for Fault Tolerant System Recovery After Its Failure. In: *Počítačové architektury & diagnostika 2014*. Malá Skála: Technická univerzita v Liberci, 2014, s. 111-116. ISBN 978-80-7494-027-9. (90%)

9. Szurman, K.; Kaštil, J.; Straka, M.; Kotásek, Z.: Fault Tolerant CAN Bus Control System Implemented into FPGA. In: *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE Computer Society, Karlovy Vary, 2013, s. 289-292, ISBN 978-1-4673-1185-4. (25%)

10. Szurman, K.: Fault Tolerant CAN Bus Control System Implemented into FPGA and its synchronization after failure and recovery. In: *Počítačové architektury & diagnostika 2013*. Plzeň: Západočeská univerzita v Plzni, 2013, s. 21-26. ISBN 978-80-261-0270-0. (90%)

Publikace autora citované jinými autory

- 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*, IEEE Computer Society, Káhira, EG, 2014, s. 231-236, ISBN 978-1-4799-6594-6.

  - Leite, A.; Pinto, A.; Matos, A.: A Safety Monitoring Model for a Faulty Mobile Robot. In: *Robotics*, volume 7, issue 3, article 32, červen, 2018, ISSN: 22186581.
  - Leite, A.: Safety features for unmanned maritime vehicles. Diplomová práce, Faculty of Engineering of the University of Porto, Porto, 2018.

- 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*, IEEE Computer Society, Verona, IT, 2014, s. 704-707, ISBN 978-1-4799-5793-4.

  - Büscher, R.: Architektur zur Unterstützung von Redundanzkonzepten auf FPGAs miels Laufzeitrekonfiguration. Diplomová práce, Faculty of Engineering and Computer Science, Hamburg University of Applied Sciences, Hamburg, 2017.

- Szurman, K.; Kaštil, J.; Straka, M.; Kotásek, Z.: Fault Tolerant CAN Bus Control System Implemented into FPGA. In: *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE Computer Society, Karlovy Vary, 2013, s. 289-292, ISBN 978-1-4673-1185-4.

  - Duman, M.: UNIBUS: a universal hardware architecture for serial bus interfaces with real-time support. Diplomová práce. Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, 2015.
  - Tremblay, J.-P.: Validation matérielle d'une architecture générique de réseaux avioniques basée sur une gestion modulaire de la redondance. Disertační práce, Polytechnique Montréal, Montreál, 2016.
  - Bhadrawat, A.-M.; Sharma, S.: DMR Based CAN Bus Control System Implemented Into FPGA, In: *International journal of scientific progress and research (IJSPR)*, Volume-04, Number - 01, 2014, ISSN: 23494689.

- Szurman, K; Kotásek, Z.: State Synchronization of Faulty Soft Core Processors in Reconfigurable TMR Architecture. In: *Počítačové architektúry & diagnostika 2017*. Smolenice: Slovenská technická univerzita v Bratislavě, 2017, s. 51-54. ISBN 978-80-972784-0-3.

  - Metinger, J.-G.: Mitigação de Erros em Cadeias de Medições de Temperatura Utilizando Interfaces Analógico-Digitais com Redundância e Diversidade. Bakalářská práce, Universidade Federal do Rio Grande do Sul. Escola de Engenharia, Porto Alegre, 2019.

Curriculum vitae

Vzdělání

2012 – *	Doktorský studijní program Výpočetní technika a informatika, FIT VUT v Brně. Školitel: Doc. Ing. Zdeňek Kotásek CSc. Státní závěrečná zkouška byla složena v roce 2015.
2009 – 2012	Magisterský studijní program Informační technologie, obor Počítačové a vestavěné systémy, FIT VUT v Brně.
2006 – 2009	Bakalářský studijní program Informační technologie, FIT VUT v Brně.
2002 – 2006	Obor Elektronické počítačové systémy, SOŠ a SOU Elektrotechnické, Lipník nad Bečvou.

Výzkumné projekty

2020 – *	Návrh, optimalizace a evaluace aplikačně specifických počítačových systémů, VUT v Brně, FIT-S-20-6309, spoluřešitel.
2017 – 2019	Pokročilé paralelní a vestavěné počítačové systémy, VUT v Brně, FIT-S-17-3994, spoluřešitel.
2014 – 2016	Architektury paralelních a vestavěných počítačových systémů, VUT v Brně, FIT-S-14-2297, spoluřešitel.
2012 – 2015	Metodiky pro návrh systémů odolných proti poruchám do rekonfigurovatelných architektur - vývoj, implementace a verifikace, MŠMT ČR - COST CZ (2011-2017), LD12036, spoluřešitel.

Odborná praxe

2018 – *	Embedded Software Developer Aerospace, TTTech, Vídeň.
2015 – 2018	Software Design Engineer, Honeywell Aerospace, Honeywell Technology Solutions, Brno.
2008 – 2015	Embedded Software Developer, UNIS a.s., Division Aerospace and Advanced Control, Brno.
2007	Software Developer, RADAS s.r.o., Brno.