VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY

DEPARTMENT OF MICROELECTRONICS

ČÍSLICOVÝ MODEL ARCHITEKTURY NETWORK ON CHIP PRO DIAGNOSTICKÉ ÚČELY

A DIGITAL NETWORK-ON-CHIP ARCHITECTURE MODEL FOR DIAGNOSTICS

BAKALÁŘSKÁ PRÁCE

SEMESTRAL THESIS

AUTOR PRÁCE — Marek Valachovič

AUTHOR

VEDOUCÍ PRÁCE — Ing. Martin Šťáva, Ph.D.

SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Mikroelektronika a technologie**

Ústav mikroelektroniky

Student: Marek Valachovič ***ID:*** 203367 ***Ročník:*** 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Číslicový model architektury Network on Chip pro diagnostické účely

POKYNY PRO VYPRACOVÁNÍ:

Zvolte a ve vhodném jazyku popište číslicový model architektury Network on Chip (NoC), který bude sloužit pro ověřování diagnostických postupů za účelem zlepšení výkonnostních parametrů dané architektury v případě výskytu poruchy. Vhodným popisným jazykem jsou např. VHDL, SystemC (C/C++), Handel-C apod.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání:3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: Ing. Martin Šťáva, Ph.D.

doc. Ing. Jiří Háze, Ph.D.
předseda rady studijního programu

ABSTRAKT

Z důvodu stále se zvyšující se integrace na čipu je komunikace pomocí sběrnice čím dál méně výhodná. Z toho důvodu vznikl Network on Chip (NoC) jakožto řešení tohoto problému. V této práci jsou popsány základní bloky, ze kterých se NoC architektura skládá. Dále je podrobně popsán model této architektury s názvem Bonfire, jsou rozebrány jak funkční bloky, ze kterých je směrovač vytvořen tak i síťové rozhraní které spojuje tento směrovač s výkonnou jednotkou.

KLÍČOVÁ SLOVA

Network on Chip, směrovač, NoC Bonfire, řízení toku dat, výkonost sítě

ABSTRACT

Due to increasing integration on the chip, bus structure for on-chip comunication is less and less advantageous. For this reason, Network on Chip (NoC) was created as a solution to this problém. In this work, the basic blocks of the NoC architecture are described. The model of this architecture called Nonfire is described in detail, both function blocks, from which the router is created, and the network Interface Connecting this router with the Processing Element.

KEYWORDS

Network on Chip, Router, NoC Bonfire, Flow Control, network performance

VALACHOVIČ, M. Číslicový model architektury Network on Chip pro diagnostické účely. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky, 2019. 42s., 0 s. příloh. Semestrální práce. Vedoucí práce: Ing. Martin Šťáva, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Marek Valachovič
VUT ID studenta:	203367
Typ práce:	Bakalářská práce
Akademický rok:	2019/20
Téma závěrečné práce:	Číslicový model architektury Network on Chip pro diagnostické účely

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: **13. prosince 2019**

.....
Podpis autora

PODĚKOVÁNÍ

Tímto bych velice rád poděkoval mému vedoucímu bakalářské práce Ing. Martinu Šťávovi Ph.D. za vlídnou a ochotnou pomoc s mými problémy spojenými s toto prací, a to jak časových, tak technických, také za jeho přístup během specifických problémů způsobených neočekávatelnými okolnostmi.

OBSAH

ÚVOD

Při zvyšující se hustotě integrace VLSI (Very Large Scale Integration) designů a vzrůstající složitosti jednotlivých komponent systému. Z důvodu vzrůstající složitosti a nárokům na co největší pracovní frekvenci při co nejmenším čase návrhu, vznikla potřeba integrovat různorodé funkční prvky na jediný čip. Například architektury MPSoC (MultiProcessor Systém on Chip) nebo CMP (Chip MultiProcessor) které používají sběrnicovou topologii pro komunikaci na čipu. Komunikační systém založený na sběrnicové topologii, která má nedostatky ve stabilitě a předvídatelnosti, není schopena udržet krok se zvyšujícími se požadavky budoucích systémů na čipu (Soc) a to ve výkonu, napájení, časování, stabilitě a dalších. Pro dosažení těchto požadavků se vytvořila strukturovaná a škálovatelná propojovací architektura s názvem NoC (Network on Chip), která má zmírnit složité problémy s komunikací na čipu. [3]

Tato práce se zabývá úpravou vybraného modelu architektury Network on Chip, který bude sloužit pro ověřování diagnostických postupů pro zvýšení výkonosti této architektury.

Nejprve jsou v této práci popsány obecné části, z kterých se NoC architektura skládá a jsou popsány různé způsoby řešení řízení toku dat a různé typy topologií. V další části je popis vybraného modelu, který je popsán v jazyce VHDL a byl vytvořen v rámci projektu Bonfire. V tomto popise je podrobně vysvětlen chod všech jeho částí, z kterých je směrovač (angl. Router) složen.

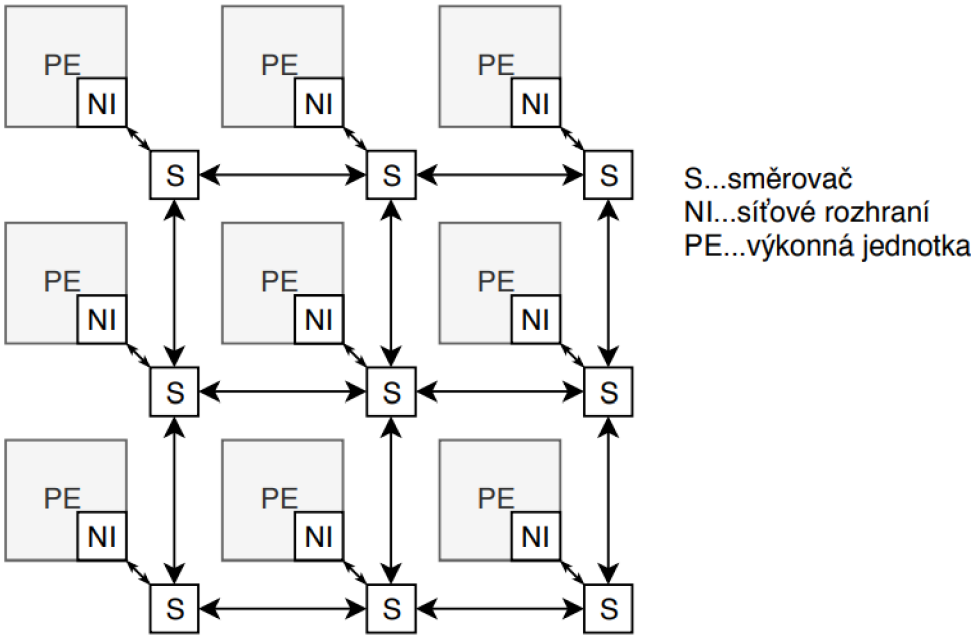
Dále jsou probrány prvky, které mají za úkol simulovat okolní prostředí s vyskytující se příčinou vzniku možné poruchy, a to, jak vliv mezi nimi samotnými, tak i důsledky mající na testovací architekturu. Na závěr je probrána lokalizace a identifikace poruchy jak mezi jednotlivými směrovači, tak i v rámci směrovače a změny projevující se na síť při jejích implementaci.

1 NETWORK ON CHIP

V následující kapitole bude popsáno, z jakých hlavních bloků se skládá tato architektura a krátkého popisu těchto bloků. Dále jsou zde popsány způsoby posílání dat mezi jednotlivými směrovači a různé typy topologií a jejich parametry.

1.1 Základní bloky NoC

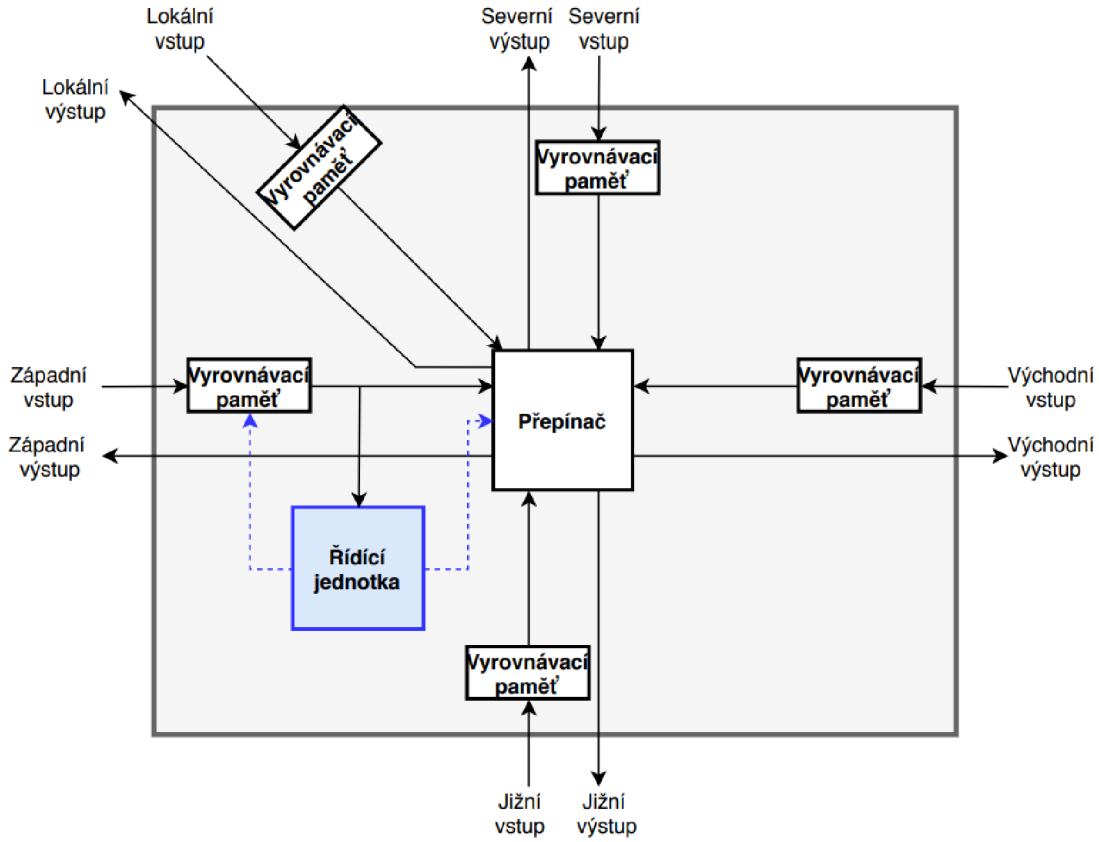
Architektura Network on Chip lze rozdělit na tří základní bloky, a to na směrovače (angl. Router) síťové rozhraní (angl. Network interface; NI) a výkonné jednotky (angl. processing element; PE). Směrovače jsou mezi sebou různě propojeny (obr. 1) a posílají si mezi sebou data. Tyto data mají svůj počátek ve výkonné jednotce, která může být například jádro procesoru, ta pošle datový obsah, tedy data, která jsou určené pro příjemce, síťovému rozhraní, které tyto surová data přetvoří do takzvaných flitů o předem stanovené délce. Flit je základní datové uskupení, flit může být hlavičkový, koncový nebo jeden z mnoha středových flitů které jsou mezi hlavičkovým a koncovým flitem. Tyto flity jsou po jejich vytvoření poslané ze síťového rozhraní do směrovače, se kterým je spojeno. Uskupení flitů je poté směrováno k jejich cílové výkonné jednotce pomocí postupného přeposílání z jednoho směrovače do sousedního, toto bude více vysvětleno v kapitole pojednávající o řízení toku dat.



Obr. 1: Ukázka možného vzájemného zapojení třech základních bloků architektury NoC [3]

1.1.1 Směrovač

V mřížkovém rozložení, které je zobrazeno na obr. 1, má každý směrovač pět vstupů a výstupů odpovídající severnímu, jižnímu, východnímu a západnímu směru a lokální který je připojen přes síťové rozhraní k výkonné jednotce. Každý vstup/výstup je spojený k sousedícímu směrovači (nebo k NI v případě lokálních) skrze několik fyzických vodičů, jejich počet odpovídá bitové velikosti flitů. Pokud v nějakém směru nemá směrovač sousedící směrovač, pak v tomto směru nejsou u mřížkové topologie vstupy a výstupy směrovače připojeny. Funkce směrovačů je nasměrovat flity přicházející z některého ze vstupů na správný výstup tak aby flity směřovali směrem k cílové výkonné jednotce. Pro správný chod je směrovač vybaven vyrovnávací pamětí na každém jeho vstupu a řídící jednotku ovládající přepínač o velikosti 5 × 5 určený k přesměrování datového toku správným směrem, toto je zobrazeno na (obr. 2). [3]

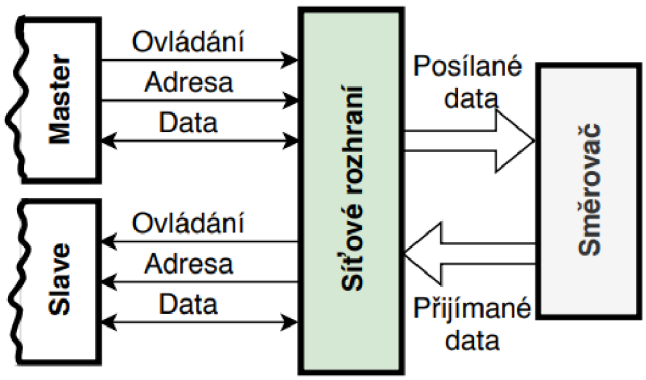


Obr. 2: Blokové zobrazení klasického směrovače použitého v mřížkové topologii [3]

Posílané flity se seskupují do takzvaných paketů. Paket je složen z jednoho hlavičkového flitu, obvykle několik středových flitů a jednoho koncového flitu, přesně v tomto pořadí. V hlavičkovém flitu bývají informace, podle kterých řídící jednotka směrovače určí, kterým směrem bude poslán celý paket. [3]

1.1.2 Síťové rozhraní (NI)

Síťové rozhraní spojuje směrovač s výkonnou jednotkou a je odpovědné jak za posílaní, tak i přijímání paketů do a ze směrovače. Po úpravě přijatých paketů je přepošle výkonné jednotce. Například pokud je NI zapojeno tak jak na obr. 3 tak v části NI, která je propojená s master se implementuje slave rozhraní a část propojená se slave se chová jako master. Propojení se směrovačem obsahuje oddělené spoje určené pro posílaní a přijímání dat, které na sobě nejsou závisle a přenáší pakety podle pravidel řízení toku dat. [5]



Obr. 3: Ukázka možného zapojení síťového rozhraní se směrovačem a výkonnou jednotkou [5]

1.2 Řízení toku dat

Řízení toku dat (angl. se zabývá posíláním dat mezi dvěma směrovači, které nemusejí být sousední. Jedná se tedy o způsob přeposílání paketů mezi směrovači takovým způsobem, aby se data dostali do cílové lokace. Řízení toku dat je velice závislé na topologii NoC architektury. Jsou dva základní způsoby posílání dat, ty budou níže popsány. [5]

1.2.1 Přepojování zpráv (angl. Store and Forward)

Pro tento typ řízení toku dat se celý poslaný paket ukládá ve vyrovnávací paměti směrovače. Z toho důvodu je nutné mít velkou velikost těchto pamětí. Dokud nejsou všechny flity uloženy nemůže se začít s jejich přeposíláním. [3]

1.2.2 Přeposílaní červí dírou (angl. Wormhole)

Při tomto typu přeposílání směrovač po přijetí hlavičkového flitu z něj ihned zjistí informace o cílové lokaci a pokud není komunikační linka již zaneprázdněná zahájí přenos dat. Z toho důvodu není potřeba mít velké vyrovnávací paměti. [3]

1.3 Parametry

Síťové topologie mají různé parametry, které se měří a na jejich základě se tyto topologie porovnávají. Níže jsou uvedeny používané parametry s jejich krátkým popisem.

1.3.1 Střední doba mezi chybami (angl. Mean Time Between Failures; MTBF)

Jedná se o dobu, po kterou by zařízení mělo vydržet v bezporuchovém stavu. Určuje se u zařízení, které jsou určena k opravě v případě poruchy. Pomocí její hodnoty se ohodnocuje spolehlivost zařízení. Vypočítává se ze statistického vyhodnocení poruchovosti zařízení. [2]

1.3.2 Latence (angl. Latency)

Jedná se o dobu, kterou paket potřebuje k dosažení cíle od svého zdroje. Latence se v síti liší podle přetížení a zatížení provozu. Je jedním z výkonnostních parametrů sítě NoC za různých provozních podmínek. [2]

1.3.3 Propustnost (angl. Throughput)

Propustnost sítě je definovaná jako rychlost dat zpracovávaných sítí v daném čase, běžně se udává v počtu bitů za sekundu. Ideální podmínkou v síti je maximální propustnost, kdy všechny kanály přenášení data okamžitě. Propustnost záleží také na dalších faktorech jako jsou řízení toku dat a směrovací algoritmy. [2]

1.3.4 Energetická spotřeba (angl. Energy Consumption)

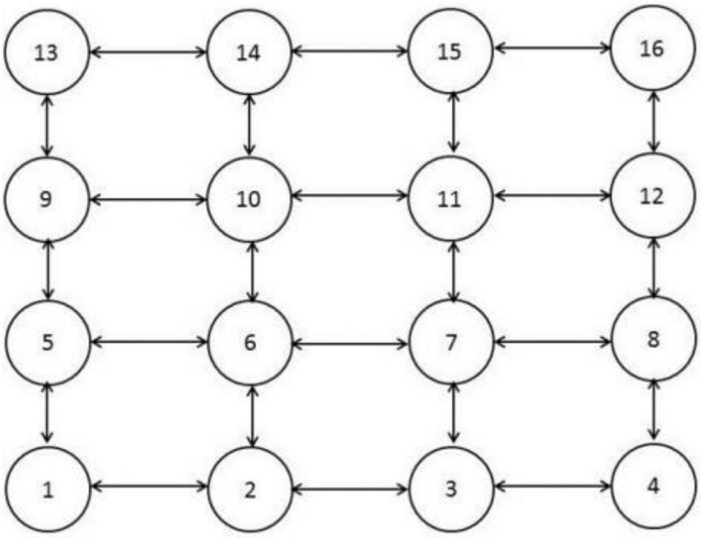
Přenos paketů po síti spotřebovává energii. Topologie s více spoji mají tendenci spotřebovávat více energie. Dalším zdrojem spotřeby energie jsou vyrovnávací paměti ve směrovači. Vzrůstající teploty mohou zvýšit energetickou spotřebu a vést tak k výraznějšímu opotřebení čipu. [2]

1.4 Typy topologii

NoC systémy mohou být implementovány mnoha různými topologiemi. Výběr topologie závisí na požadavcích a parametrech, jakou jsou třeba složitost implementace, rozměrová velikost a směrovací algoritmus. Níže jsou uvedeny příklady používaných topologií. [2]

1.4.1 Mřížková topologie (angl. Mesh)

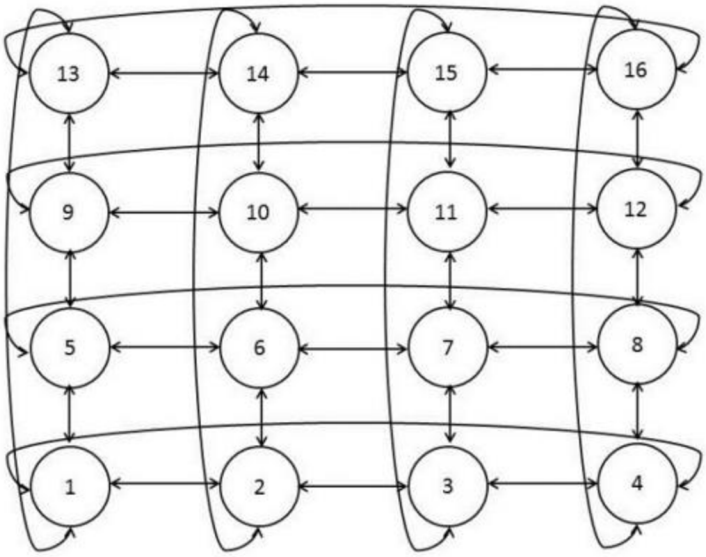
Mřížková topologie je nejpoužívanější a nejzákladnějším typem topologie s velmi jednoduchým směrovacím algoritmem. Je snadno implementovatelná a je škálovatelná. Nevýhodou je velký počet potřebných přeskoků dat při jejich posílání na větší vzdálenost. Například pro mřížku o rozměrech 4 × 4 (obr. 4) je potřeba šest skoků, aby se data dostali ze směrovače 1 do 16. Pro směrování lze použít směrovací algoritmus, který směruje data na základě rozdílu v souřadnicích x a y. [2]



Obr. 4: Ukázka mřížkové topologie NoC architektury o velikosti 4 × 4 [2]

1.4.2 Toroidní topologie (angl. Torus)

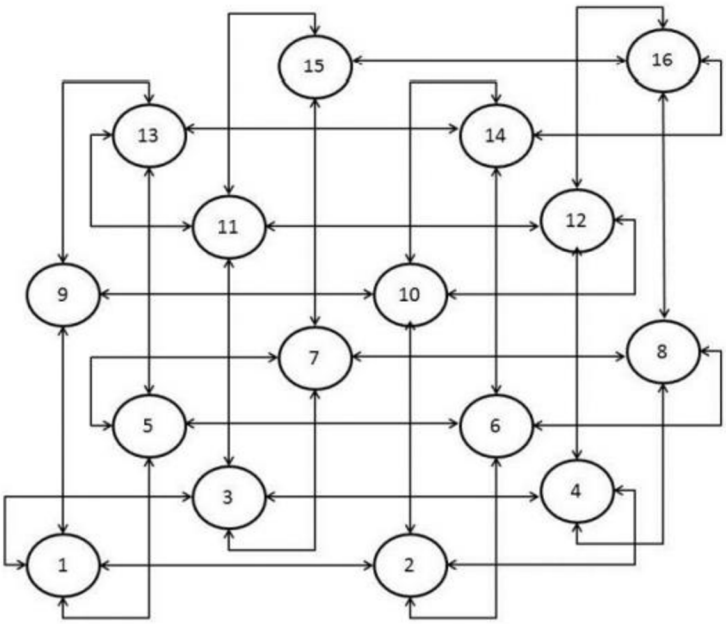
Toroidní topologie vychází z mřížkové, ale na rozdíl od ní spojuje okrajové směrovače s protějšími ve stejném řádku nebo sloupci (obr. 5). Díky těmto spojení se tak zmenší nejdelší datová cesta a sníží se tak maximální počet přeskoků dat. Pro toroidní síť o velikosti 4 × 4 je maximální počet přeskoků roven čtyřem což je lepší než u mřížkové sítě o stejné velikosti. [2]



Obr. 5: Ukázka toroidní topologie NoC architektury o velikosti 4 × 4 [2]

1.4.3 Přeskládaná toroidní topologie (angl. Folded Torus)

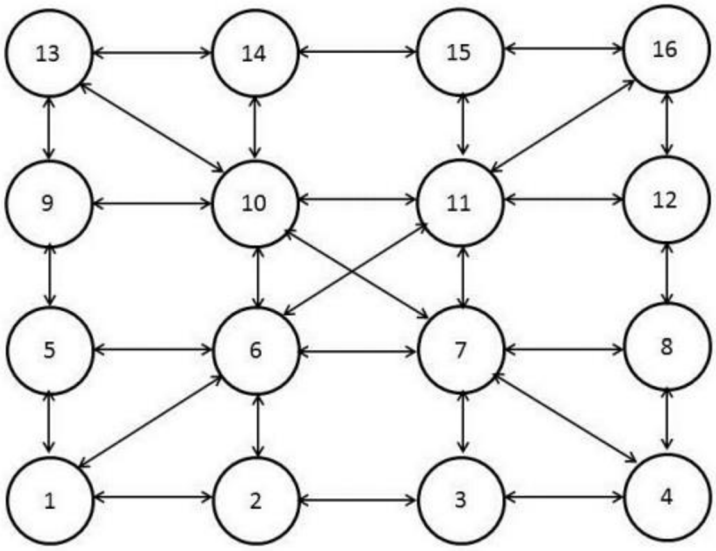
Jedná se o další variantu toroidní topologie s tím rozdílem, že má kratší délku spojů, což pomáhá snížit čas potřebný k přenosu paketu po spojích (obr. 6). Kratší délka spojení také přispívá ke zmenšení propojovací oblasti potřebné pro implementaci. Tato topologie má také větší rozmanitost cest než klasická Toroidní topologie a je odolnější vůči chybám. [2]



Obr. 6: Ukázka Přeskládané toroidní topologie NoC architektury o velikosti 4 × 4 [2]

1.4.4 Diagonálně-mřížková topologie (angl. Xmesh)

Diagonálně-mřížková topologie je vylepšení klasické mřížkové topologie s tím rozdílem, že přidává spoje mezi směrovači i do kolmého směru, ale pouze na hlavních diagonálách (obr. 7). Tímto se nejdelší datová cesta sníží na polovinu a sníží se tak maximální počet přeskoků dat. Přidané diagonální spoje přispívají k zvětšení potřebné oblasti. [2]



Obr. 7: Ukázka diagonálně-mřížkové topologie NoC architektury o velikosti 4 × 4 [2]

2 NOC MODEL BONFIRE

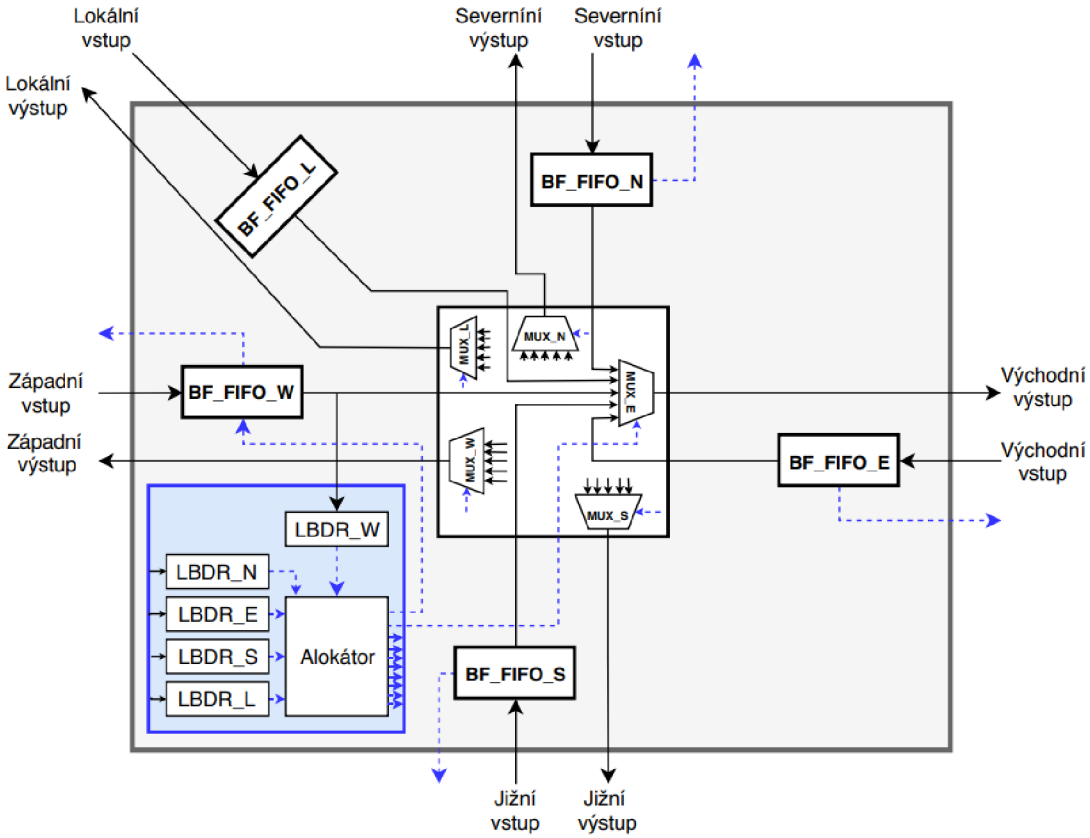
V této kapitole je popsán model architektury NoC který je pojmenován Bonfire. Tento model byl zvolen k implementaci různých vylepšení, aby se zvýšila výkonnost této architektury, z toho důvodu je popis velice podrobný.

2.1 Směrovač

Směrovač v tomto projektu má 5 vstupů a 5 výstupů a je určen pro použití ve 2D mřížkové topologii. Používá řízení toku dat způsobem přeposílaní červí dírou. Posílaný paket se musí skládat z jednoho hlavičkového flitu (povinné), z žádného nebo více středových flitů (volitelné) a jednoho koncového flitu (povinné). Po přijetí flitu směrovačem trvá 3 časové cykli, než flit směrovač opustí. Směrování je realizováno jako logické, z toho důvodu není použita směrovací tabulka. Směrovač obsahuje pro každý vstup vyrovnávací paměť typu FIFO, které jsou velké 4 × 32 bitů. Pro každý vstup dále obsahuje blok pro logicky založené směrování (LBDR), který určuje směr, kterým se data vydají. Směrovač obsahuje alokátor, který se stará, aby nedošlo ke kolizím a pro každý výstup obsahuje směrovač přepínač (XBAR) který podle pokynů alokátoru propojí svůj výstup s výstupem vyrovnávací paměti ve které jsou uložena data. Funkce jednotlivých bloků jsou popsány níže společně s popisem formátu posílaných dat a síťového rozhraní. [2]

Pro popis tohoto směrovače se budou používat pro některé bloky jiné názvy, a to z toho důvodu, neboť některé názvy jsou nepřesné či zavádějící. Název jednotlivých přepínačů je v tomto projektu XBAR (crossbar switch), což implikuje, že tyto přepínače propojují jeden ze svých pěti výstupů na jeden z vícero výstupů. Tyto přepínače mají ovšem pouze jeden výstup, jsou tedy multiplexory, a až po spojení všech pěti do jednoho bloku by vzniknul XBAR. V této práci tedy bude blok XBAR nazýván jako multiplexor (MUX). Vyrovnávací paměti pojmenované FIFO zpracovávají a dále posílají jeden z řídících signálů, neboť tuto funkci paměti typu FIFO nemají vykonávat bude jejich název taktéž mírně pozměněn, a to na BF_FIFO. Jednotlivé bloky stejného typu jsou v projektu Bonfire odlišeny pomocí prvního písmena anglického názvu směru, ve kterém se daný blok nachází. Toto je v tomto popise zachováno. [2]

Na vyobrazení blokového schématu směrovače (obr. 8) si lze povšimnout, jak jsou spolu jednotlivé bloky propojeny. Pro zjednodušení jsou plně zobrazeny signály pouze pro západní vstup a východní výstup. Toto blokové schéma je nakresleno tak aby byly patrné podobnosti s obecným blokových schématem směrovače pro NoC architekturu (obr. 2). [2]



Obr. 8: Blokové schéma směrovače z projektu Bonfire [1] [2]

2.2 Formát posílaných dat

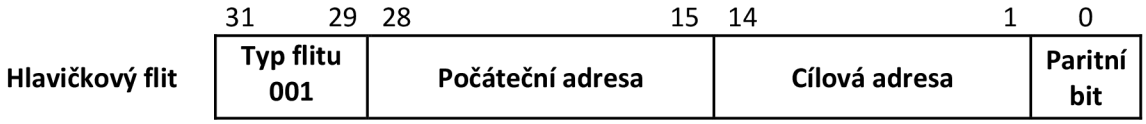
Data se posílají v paketech, které se skládají z flitů. Velikost jednotlivých flitů je 32 bitů. Z toho první 3 bity jsou typové, tedy určují, o jaký flit se jedná, flit může být hlavičkový, středový nebo koncový, o jaký typ se jedná určí síťové rozhraní. Poslední byt je vždy paritní bit, který je vypočítán síťovým rozhraním. [2]

Tab. 1: Typy flitů podle hodnoty prvních tří bitů [2]

Hodnota	Typ flitu
001	Hlavičkový
010	Středový
100	Koncový

2.2.1 Hlavičkový flit

Hlavičkový flit je v paketu vždy jen jednou, a to na začátku. Podle něj směrovač vyhodnocuje směr, kterým celý paket postupně pošle. Toto vyhodnocení se provede porovnáním aktuální adresy s adresou cílovou. Celkový obsah hlavičkového flitu i s bitovým rozložením je na (obr. 9).

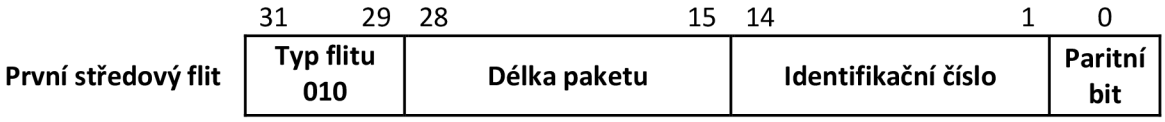


Obr. 9: Bitové rozložení v hlavičkovém flitu [2]

Počáteční adresa je vytvořena automaticky síťovým rozhraním a cílová adresa je vytvořena během prvního zápisu výkonné jednotky. [1] [2]

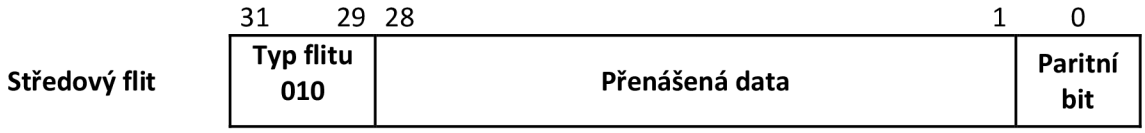
2.2.2 Středový flit

Středových flitů je v paketu více. Rozdělují se na dva typy, a to na první středový flit a na ostatní středové flity. První středový flit obsahuje informace o délce paketu a jeho identifikační číslo, je možné vidět na



Obr. 10: Bitové rozložení v prvním středovém flitu [2]

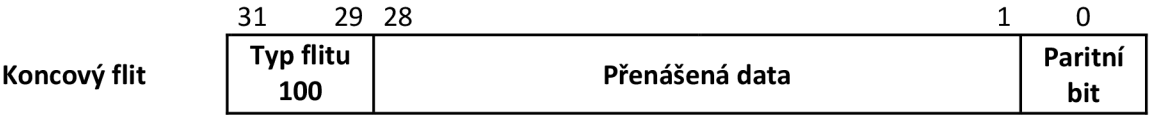
Identifikační číslo paketu je určeno a zapsáno síťovým rozhraním a délka paketu je zapsána během druhého zápisu výkonné jednotky. Ostatní středové flity obsahují přenášená data, které jsou zapsány výkonnou jednotkou. Ostatní středové flity jsou zobrazeny na obr 11.



Obr. 11: Bitové rozložení v ostatních středových flitech [2]

2.2.3 Koncový flit

Koncový flit se v posílaném paketu nachází vždy pouze jednou, a to na konci. Koncový flit má stejnou strukturu jako středové flity a jeho účelem je, mimo přenášení dat, sdělit směrovači informaci, že přenos paketu může skončit po jeho poslaní. Koncový flit je zobrazen na obr. 12.



Obr. 12: Bitové rozložení v koncovém flitu [2]

2.3 Bloky směrovače

V této kapitole budou rozebírány jednotlivé bloky, z kterých je složen směrovač z projektu Bonfire.

2.3.1 BF_FIFO

Vyrovnávací paměti na vstupech směrovače jsou paměti typu FIFO (First In First Out), to je typ paměti, u které data odcházejí ve stejném pořadí, v jakém do paměti přišly. Zde je vyrovnávací paměť použita jako část datové cesty směrovače, slouží k ukládání flitů což je spojeno se způsobem řízení toku dat, které je prováděno typem červích děr. Proto není potřeba aby se do paměti dal uložit celý paket. Velikost této paměti je taková, aby zvládla pojmout čtyři flity které mají velikost 32 bitů. V tomto provedení směrovače je celkem pět takovýchto vyrovnávacích pamětí, a to jedna pro každý jeden z jeho pěti datových vstupů. [1] [2]

Zápis dat do vyrovnávací paměti BF_FIFO probíhá prostřednictvím 32bitového vstupního signálu *RX*. Do kterého paměťového bloku *FIFO_MEM_i* (i=1, 2, 3, 4), ze čtyř možných, se tyto data uloží určuje ukazatel zápisu (*write_pointer*). Tento ukazatel je 4bitový a vždy jen jeden jeho bit je v log. 1, obsahuje tedy kód 1 z 4. Po přijetí log. 1 z předchozího směrovače nebo síťového rozhraní prostřednictvím signálu *valid_in*, který potvrzuje platnost přijatých dat, a skutečnosti, že ještě nejsou zaplněna všechna paměťová místa, obsah ukazatele zápisu zarotuje o jedna doleva. O informaci že nejsou zaplněny všechny paměťové bloky se stará signál *full,* který porovnává hodnoty ukazatele zápisu s ukazatelem čtení. [1] [2]

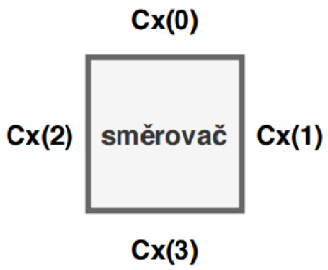
Pokud alokátor povolí čtení pomocí jednoho ze signálů *read_en_i* (i = N, E, W, S, L) a paměťové bloky nejsou prázdné (*empty* = 0), nastaví se signál *read_en* do log. 1 a následně i výstupní signál *credit_out*. Tento signál informuje alokátor směrovače nebo síťové rozhraní, ze kterého se budou data zapisovat o tom, že se uvolňuje místo pro další zápis dat. Data z vybraného paměťového bloku jsou vedeny ven z vyrovnávací paměti částečně do bloku LBDR (Logic-Based Distributed Routing) a kompletně do multiplexoru. Z kterého paměťového bloku se budou data číst určuje ukazatel čtení (*read_pointer*), který je taktéž jako ukazatel zápisu v kódu 1 z 4 a vždy při povolení čtení svou hodnotu zarotuje doleva. Jestli paměťové bloky nejsou prázdné určuje signál *empty*, který porovnává hodnoty ukazatelů čtení a zápisu. Signál *empty* je vyveden z vyrovnávací paměti BF_FIFO do dalších bloků směrovače, a to konkrétně do LBDR a alokátoru. [1] [2]

Vyrovnávací paměť BF_FIFO může být asynchronně restartována pomocí signálu *rst*, restartování tedy není závislé na synchronizačním signálu *clk*. Restartování proběhne tehdy když se signál *rst* nastaví do logické 0. Během restartu se hodnota ukazatele zápisu a ukazatele čtení nastaví na “0001“, ty tak začnou ukazovat na první

paměťový blok. Dále se všechny pozice ve všech čtyřech paměťových blocích nastaví na logickou 0, stejně tak se i signál *credit_out* nastaví do log. 0. [1] [2]

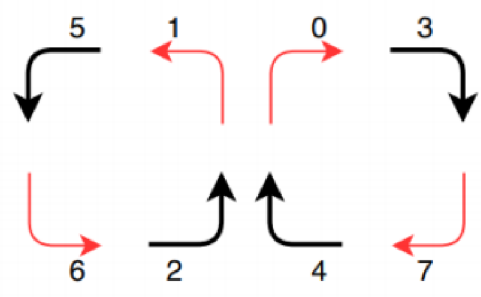
2.3.2 LBDR

Blok LBDR (Logic-Based Distributed Routing) se podílí na řídící části směrovače, a to tak že určuje kterým výstupem data potečou. Směrovač obsahuje celkem pětkrát LBDR blok, a to jeden pro každou vyrovnávací paměť BF_FIFO. Data můžou směřovat do jednoho z výstupů které jsou propojeny se sousedícími směrovači nebo do lokální části spojené se síťovým rozhraním a výkonnou jednotkou. Data blokem LBDR neprochází, LBDR pouze sleduje malou část dat, pomocí které vytvoří žádost o povolení čtení, která je mířena do alokátoru. LBDR obsahuje dva konfigurační signály. Signál propojení (angl. connectivity) *Cx* který je 4bitový a popisuje, v kterých směrech je směrovač propojen, každý bit určuje jeden směr, pokud je daný bit v log. 1 je směrovač v daném směru propojen. [2]



Obr. 13: Zobrazení přiřazení jednotlivých bitů signálu Cx ke konkrétním směrům [2]

Druhý signál je 8bitový směrovací *Rxy*, jednotlivé bity tohoto signálu určují, které způsoby zatočení jsou povolené a které zakázané. Tohoto se využije v případě, ve kterém by flit změnil svůj hlavní směr při průchodu směrovačem. Pokud by byla hodnota signálu *Rxy* = 00111100 získali bychom povolené a zakázané zatočení která jsou zobrazena na obr. 14. Černé šipky jsou povolená zatočení a červené jsou zakázané. Číslice udávají pořadové číslo jednotlivých bitů. [2]



Obr. 14: Přiřazení jednotlivých bitů signálu Rxy k jednotlivým způsobům směny směru toku dat [2]

Blok LBDR sleduje 17 bitů z datového výstupu bloku BF_FIFO, konkrétně tedy

první 3 bity (typové) a ty s pořadovým číslem mezi 1 až 14 na kterých je, v případě že se jedná o hlavičkový flit, informace o cílové lokaci. LBDR porovnává svoji adresu s adresou cílové lokace z čehož určí směr kterým by se měli data poslat, pomocí nastavení log. 1 do jednoho či dvou ze signálů *N1* (sever), *E1* (východ), *W1* (západ), *S1* (jih). [1] [2]

V případě, že sledovaná data z bloku BF_FIFO budou hlavičkový flit a signál *empty*, přivedený taktéž z bloku BF_FIFO, bude v log. 0, tak LBDR začne porovnávat signály určující směr. V případě, že je směr jednoznačný, tedy pouze jeden signál určující směr je v log. 1, záleží jen na hodnotě signálu *Cx*, pokud je jeho hodnota log. 1, tak LBDR vytvoří žádost o povolení čtení v daném směru. Pokud je směr nejednoznačný, tedy jsou dva signály určující směr v logické 1, pak směr určí hodnota signálu *Rxy*, daný směr opět musí být povolený signálem *Cx*. V případě, že ani jeden ze směrových signálů není v logické 1, aktuální adresa je shodná s cílovou, nastaví se žádost o povolení zápisu do síťového rozhraní, tato žádost nezávisí na signálech *Rxy* a *Cx*. [1][2]

Takto nastavená žádost setrvává beze změny až do momentu, kdy sledovaná data z bloku FIFO budou koncový flit, pak jestli je jakékoliv povolení čtení aktivní, signál *grants* je v log. 1, tak se žádost zruší. [1] [2]

Blok LBDR obsahuje asynchronní restart pomocí signálu *reset*, restartování proběhne při jeho nastavení do log. 0. Při restartování se zruší všechny žádosti o povolení čtení. [1] [2]

2.3.3 Multiplexor

Multiplexor, v projektu Bonfire pojmenovaný jako XBAR, je součástí datové cesty směrovače, žádné řídící signály skrze něj neprocházejí, každopádně je ovládaný signály z alokátoru, který slouží i jako kontrolní část směrovače. multiplexorů je celkem pět ve směrovači, a to pro každý datový výstup jeden. Každý multiplexor je spojen s datovými výstupy všech pamětí BF_FIFO. To, který datový vstup se propojí s výstupem rozhoduje signál *sel,* který je 5bitový a jeho obsah je vytvořen složením 1bitových povolovacích signálů z alokátoru, který zajišťuje že tento signál je 1 z 5. [1] [2]

Tab. 2: Hodnoty řídícího signálu pro zvolení konkrétního vstupu [1]

Hodnota	Zvolený vstup
00001	Lokální
00010	Jižní
00100	Západní
01000	Východní
ostatní	Severní

Bit, který by nastavil na výstup data pocházející se shodného směru s výstupem, je vždy v log. 0. To znamená že směrovač nikdy nemůže poslat data sám sobě ani za použití sousedního směrovače. Multiplexor je čistě kombinační obvod, neobsahuje tedy synchronizační a resetovací signál. [1] [2]

2.3.4 Alokátor

Alokátor je hlavní řídící částí směrovače, rozhoduje o tom, které žádosti o povolení přenosu dat, které jsou přivedeny z bloků LBDR, se vyhodnotí jako platné. Rozhoduje tedy o tom, které data mohou kam směřovat. Dále řeší kolize více žádostí o povolení přenosu dat do stejného výstupu směrovače a jestli je v sousedním směrovači do kterého se data mají poslat prostor pro jejich přijetí. Z toho důvodu má alokátor pro každý výstup směrovače 2bitový čítač, který při přijetí informace, že data uložené ve vyrovnávací paměti BF_FIFO sousedního směrovače se přeposlala dál, tedy se uvolnilo místo v paměti, skrze signál *credit_out,* zvýší svou aktuální hodnotu o jedna. Při přijetí žádosti o povolení z modulu arbiter_out, při kterém by se povolil přenos dat ve směru, pro který je čítač určen, se hodnota tohoto čítače sníží o jedna. Při přijetí obou signálu se jeho hodnota nezmění. Pokud je hodnota čítače nulová alokátor nepovolí přenos dat v daném směru. Aby vůbec mohlo dojít k posílání dat musí být hodnota tohoto čítače při spuštění nastavena na maximální hodnotu, neboť vyrovnávací paměti jsou prázdné. Alokátor je složen z více modulů, které jsou níže popsány. [1] [2]

Prvním takovým to modulem je arbiter_in. Tento podmodul alokátoru vyhodnocuje všechny žádosti o povolení přenosu dat z jednoho směru, tedy všechny výstupní signály *Req_i* (i = N, E, W, S, L) jednoho bloku LDBR. Z toho důvodu jich v alokátoru musí být stejný počet jako datových vstupů směrovače. Obsahuje stavový automat, ve kterém v případě více žádostí o povolení přenosu dat vybere jednu. Pokud předtím ještě nebyla vybrána žádná žádost, tedy o směru, kterým se budou data posílat nebylo rozhodnuto, je směr vybírán podle následujícího pořadí: sever, východ, západ, jih a nakonec lokální. V případě, že směr již byl určen pak toto pořadí je změněno tak aby předchozí směr měl největší prioritu. [1] [2]

Druhým podmodulem alokátoru je arbiter_out. V tomto modulu se vyhodnocují všechny žádosti o povolení přenosu dat do jednoho konkrétního směru, vybírá se tedy z kterého směru budou data poslány. Jako vstupní signály jsou zde použity výstupní signály z modulů arbiter_in a to z každého modulu právě jeden, který je pro stejný směr jako arbiter_out. Tento modul obsahuje podobný stavový automat jako modul arbiter_in, stejným způsobem se zde z více žádostí vybere jedna. Pokud je nějaký směr vybrán a jsou nějaká data která lze poslat, tedy signál *empty* který je přivedený z paměti BF_FIFO ze směru, z kterého se data mají přeposlat, je v log. 0, tak je tento povolovací signál dále zpracováván alokátorem. Alokátor tento signál použije jako jeden ze vstupů do výše uvedeného čítače, který sleduje, zda je možné data poslat. Pokud je žádost alokátorem vyhodnocena jako platná, je povolovací signál přiveden do bloků LBDR a BF_FIFO ve směru, ze kterého se data posílají. [1] [2]

Alokátor obsahuje asynchronní restart spuštěný při signálu *reset* v log. 0. Při restartování se hodnoty čítačů, kontrolující volné místa ve vyrovnávacích pamětech, nastaví na svou maximální hodnotu a stav u obou arbiterů se nastaví na výchozí. [1] [2]

2.4 Síťové rozhraní

V projektu Bonfire je i model síťového rozhraní které zajišťuje přenos dat mezi směrovačem a výkonnou jednotkou. Její funkci lze tedy rozdělit na dvě základní činnosti, a to posílaní dat z výkonné jednotky do směrovače a naopak. Síťové rozhraní obsahuje dvě paměti o velikosti 32 × 32 bitů, které slouží k uložení posílaných a přijímaných dat před jejich následovném přeposlání. Obě tyto paměti jsou řízeny pomocí dvou 5bitových ukazatelů, zápisu a čtení. Každá paměť má tyto ukazatele vlastní. Porovnání hodnot těchto ukazatelů zjistíme, zda v paměti nejsou žádná zapsaná data nebo že je paměť plně naplněna. Síťové rozhraní obsahuje asynchronní restart pomocí signálu *reset*, restart proběhne při nastavení tohoto signálu do log. 1. Při restartu se všechny vnitřní signály a paměti vynulují a 2bitový čítač hlídající místo ve vyrovnávací paměti směrovače se nastaví na svou maximální hodnotu. Způsoby, jakými tyto přeposílaní fungují jsou popsány níže. [1]

2.4.1 Posílaní dat z PE do NoC

Pokud je v log. 1 signál *enable*, který je vyveden z výkonné jednotky a povoluje síťovému rozhraní čtení jeho datových výstupů, a aktuální adresa se shoduje s vyhrazenou adresou, uloží se 8bitová data z výstupu PE do 32bitového registru *storage*. Do které části se tyto data uloží určuje 4bitový signál *write_byte_enable* který je v kódu 1 z 4 nebo nulový. Pokud tento signál není nulový nastaví do log. 1 signál *valid_data*, který v případě, že paměť pro uložení dat není zcela zaplněná (*P2N_full* = 0) povolí zápis do této paměti. Uložená data v registru *storege* se uloží do paměti NI, na které místo v paměti se data uloží určuje hodnota 5bitového signálu *P2N_FIFO_write_pointer* který pracuje jako ukazatel na místo v paměti NI. Hodnota tohoto ukazatele je zvýšena o jedna vždy při nastavení povolení čtení (*P2N_write_en* = 1), které nastane po dokončení naplnění registru *storage* 8bitovými daty z výkonné jednotky. [1]

Síťové rozhraní obsahuje stavový automat, ve kterém se vytvářejí flity posílané do směrovače. Mezi jednotlivými stavy se přechází, když je splněna podmínka toho, že v paměti jsou data, která lze poslat, tedy signál *P2N_empty* = 0, a hodnota 2bitového čítače není nulová, jedná se o stejný čítač, jaký je popsán u alokátoru, přechod mezi prvním a druhým stavem není závislí na hodnotě tohoto čítače. Pokud nejsou splněny podmínky pro změnu stavu, setrvává stavový automat stále v tom stejném, z toho plyne, že je přesně daná jejich posloupnost, která je neměnná. V prvním stavu se pouze nastaví stav následující. Ve druhém stavu se při splnění podmínek udělí povolení čtení, signál *grant* je nastaven do log. 1, a je vytvořen hlavičkový flit. Ten je vytvořen složením 3 bitů které určují typ flitu, tedy 001, dvou 7bitových adres, které udávají aktuální adresu v osách y (prvních 7 bitů) a x (druhých 7 bitů), prvních 14 bitů z 32bitového signálu *FIFO_Data* který je výstupem paměti NI, kterých uložených 32 bitů z paměti se použije

určuje ukazatel čtení *P2N_FIFO_read_pointer,* který s každým povolením čtení svou hodnotu zvýší o jedna. Jako poslední bit flitu se použije paritní bit. Ve třetím stavu se po splnění podmínek povolí čtení, určí se velikost čítače délky paketu a vytvoří se první středový flit. Velikost čítače délky paketu (*packet_length_counter*) je určena odečtením dvojky od hodnoty udávající velikost paketu, tu udává signál *FIFO_Data* svými 14 bity, a to konkrétně od 14. po 27. První středový flit je vytvořen složením 3 bitů určující jeho typ, tedy 010, 14bitovů určující velikost paketu, 14bitového čítače *packet_counter* který svou hodnotu zvýší o jedna při každém vytvoření koncového flitu, slouží jako identifikační číslo paketu. Jako poslední bit flitu je použit vypočítaný paritní bit. Ve čtvrtém stavu se povolí čtení, vytvoří středový flit a sníží se o jedna hodnota čítače délky paketu. Středový flit je vytvořen spojením 3bitové hodnoty určující typ flitu, tedy 010, prvních 28 bitů ze signálu *FIFO_Data*, jedná se o datový obsah, tedy o data určené pro příjemce, a paritního bitu. Dokud je hodnota čítače délky paketu větší než 2, následující stav bude opět čtvrtý, takto se vytvoří více středových flitů. Jakmile hodnota tohoto čítače klesne pod 3, přejde stavový automat do pátého stavu. V pátém stavu se povolí čtení, sníží se hodnota čítače délky paketu o jedna, vytvoří se koncový flit a zvýší se hodnota čítače *packet_counter* jehož hodnota se používá při tvorbě prvního středového flitu jako identifikační číslo paketu. Koncový flit je vytvořen stejným způsobem jako středový. Následující stav je opět první stav. [1]

Flity jsou vytvářeny přímo na výstup síťového rozhraní a rovnou ukládány a čteny směrovačem na lokálním vstupu. [1]

2.4.2 Posílaní dat z NoC do PE

Hodnoty na lokálním výstupu směrovače jsou ukládány do paměti síťového rozhraní určené pro přijatá data. Místo v paměti je vybráno pomocí hodnoty 5bitového ukazatele zápisu *N2P_write_pointer*. Pokud je v log. 1 signál *valid_in*, který je přivedený ze směrovače a potvrzuje že data jsou určena ke čtení, a paměť není zcela zaplněna (*N2P_full* = 0), tak se udělí povolení k zápisu do paměti (*N2P_write_en* = 1). Tím se hodnota ukazatele zápisu do paměti zvětší o jedna a data vložená na předchozí adresu zůstanou v paměti uložena. Pokud je aktuální adresa shodná s vyhrazenou adresou a 4bitový signál *write_byte_enable* je nulový, výkonná jednotka tedy nechce data posílat, a v paměti NI jsou uložena přijatá data (*N2P_empty* = 0), povolí se čtení paměti (*N2P_read_en* = 1) a síťové rozhraní sdělí směrovači, že pro data je v paměti místo a mohou tak být poslána, pomocí nastavení signálu *credit_out* na log. 1. Pokud je povoleno čtení paměti a minulá adresa je shodná s adresou vyhrazenou, na výstup, síťového rozhraní, který je propojen s výkonnou jednotkou, se načtou data z paměti. Která část paměti bude takto použita rozhoduje 5bitový ukazatel čtení *N2P_FIFO_read_pointer* jehož hodnota se zvýší o jedna s každým povolením čtení za předpokladu, že v paměti jsou uložená data. Pokud tyto podmínky nejsou splněny a minulá adresa je rovna příznakové adrese, na výstup síťového rozhraní je nastaven

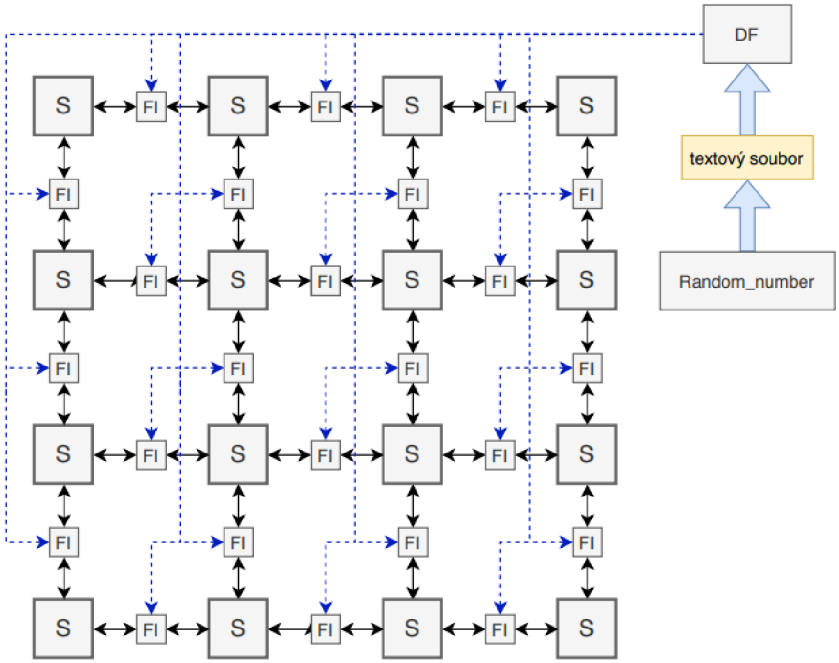
signál *flag_register* jehož 32bitová hodnota je vytvořena složením signálů *N2P_empty* a *P2N_full* k nimž je zprava připojeno 30 nul. Pokud ani toto není splněno a minulá adresa je rovna adrese čítače je na výstup nastaveno 30 nul zprava doplněnými hodnotou 2bitového čítače *counter_register*. Hodnota tohoto čítače se po zápisu hlavičkového flitu do paměti NI zvýší o jedna a při čtení koncového flitu z paměti zase snížena o jedna. [1]

3 SIMULACE OKOLÍ

V této kapitole jsou popsány přídavné bloky, které slouží jak pro vytváření a sledování poruch mezi jednotlivými směrovači tak i k tomu, aby v případě výskytu poruchy, trvalé i přechodové, byla data doručena do cílového směrovače. Dále je zde popsané testovací rozhraní, které nahrazuje NI. Pro tyto účely se využívá mřížková síť směrovačů o velikosti 4×4, která již dovoluje více alternativních cest při výskytu poruchy.

3.1 Generování poruch

Pro generování poruch a distribuci poruch slouží vzájemně nepropojené dva bloky. Blok, který vytváří data, ukládané v textovém souboru, podle kterých se poruchy generují, se nazývá Random_numbers. Blok, který tyto data využívá ke generaci poruch se nazývá FD (angl. Fault Description), jeho činností je, jak poruchu vytvořit, tak ji i distribuovat do jednoho z mnoha bloků FI (angl. Fault Injector), které propojují jednotlivé směrovače. Zapojení těchto bloků do sítě směrovačů je zobrazeno níže (obr. 15). Všechny tyto zmíněné bloky nejdou syntetizovat což nevadí, neboť jejich funkcí je jen simulovat okolní prostředí, a tedy již z jejich podstaty se v konečné architektuře sítě nevyskytnou.



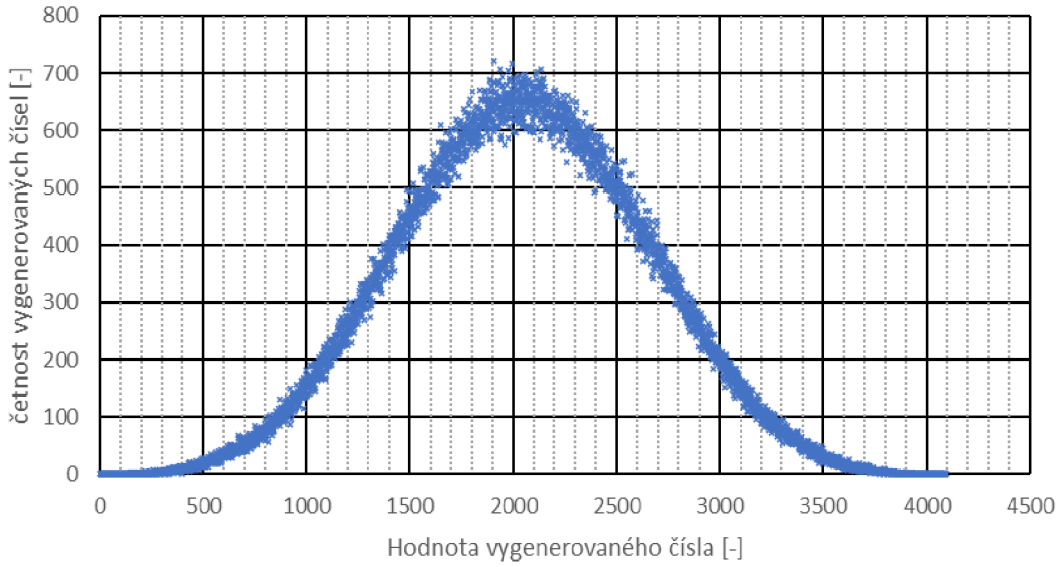
Obr. 15: Zapojení sítě směrovačů o velikosti 4×4 s bloky generujícími a poruchy

3.1.1 Blok Random_numbers (RN)

Jako základ kódu tohoto bloku je použit generátor pseudonáhodných čísel, u něhož je nepoměr pravděpodobnosti výskytu mezi jednotlivými čísli, a to takovým způsobem že četnosti jednotlivých čísel připomínají Gaussovo rozdělení. Tento generátor pro svůj chod využívá čtyř počátečních hodnot (angl. Seed) z nichž je každá o velikosti 32 bitů. Postupným kombinováním bitů v rámci každého z čísel tak i kombinováním výsledných čísel vzniká 12bitové pseudonáhodné číslo, které je vždy po třech kombinačních cyklech vyvedeno na výstup. [7]

Rozšiřující část do tohoto generátoru přidává pole o takové velikosti, aby každá číslice měla vlastní část v tomto poli. Jelikož je výstupní hodnota 12bitová je tedy počet možných čísel roven 4096 a pole tedy musí mít velikost 0 až 4095. Vždy po vygenerování pseudonáhodného čísla se jeho hodnota upraví takovým způsobem, aby výsledná hodnota byla čistě kladná, ale nezměnil se celkový charakter četností jednotlivých čísel (střed grafu četností čísel v závislosti na hodnotě čísla je posunut z 0 do 2047). Takto upravená hodnota se použije jako index výše zmíněného pole a v tomto místě se hodnota v poli zvýší o jedna. Takto se při generování pseudonáhodných čísel ukládá jejich četnost. Při každém vygenerovaném čísle se zvýší hodnota čítače o jedna, a to až do momentu kdy jeho hodnota dosáhne předem určeného maxima, které je v tomto případě stanoveno na hodnotě 10^6, poté již dalším pseudonáhodným číslům není dáván žádný význam a četnosti jednotlivých čísel uložené v poli jsou následně vypsány do textového souboru.

Tento textový soubor obsahuje pouze hodnoty bez jakéhokoliv textu, neboť takto vypsaná data se nadále snáze zpracovávají. Důvod, proč byl zvolen způsob vypsání výsledků do textového souboru je takový že celý proces vygenerování 10^6 pseudonáhodných čísel, které každé trvá tři cykly, vyžaduje 30 ms simulovaného času, které odpovídají až desítkám minut reálného času. Četnosti jednotlivých čísel jsou tedy vygenerovány nezávisle na simulaci sítě směrovačů, aby nedocházelo k zbytečnému prodlužování těchto simulací (výsledky tohoto bloku budou totiž vždy stejné při nezměněných 32bitových počátečních hodnotách).



Obr. 16: Histogram závislosti četnosti na hodnotě vygenerovaného čísla

3.1.2 Blok Fault Description (FD)

Jak z názvu vyplývá, tak tento blok hrubě popisuje nastávající chyby, které podrobněji zpracuje příslušný blok FI. Blok FD přímo pracuje s daty v textovém souboru vytvořeným blokem Random_numbers.

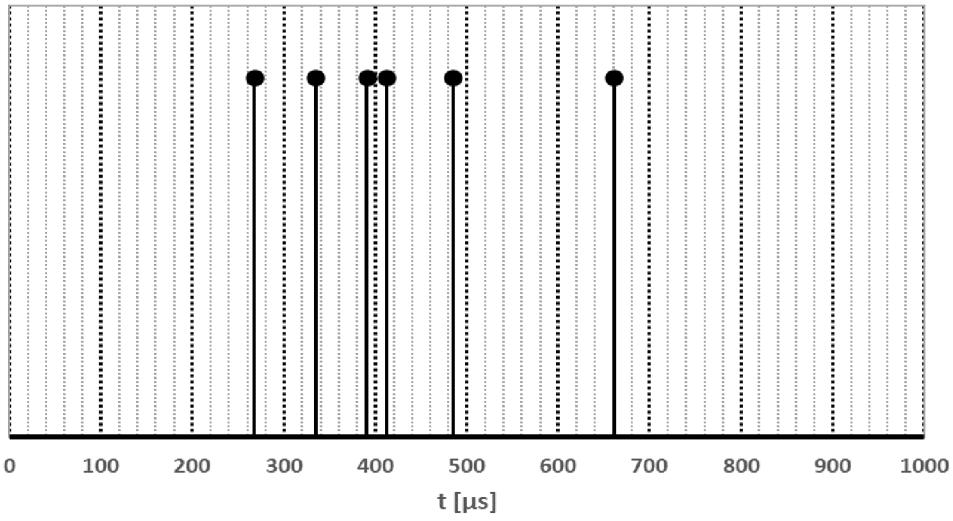
FD v inicializaci ihned po zpuštění simulace postupně přečte a uloží hodnoty z textového souboru do pole o stejných rozměrech jako pole v RN (tedy 0 až 4095). S hodinovým signálem *clk* se inkrementuje hodnota čítače (*counter_sig*) který slouží jako index pro přístup do pole s uloženými četnostmi čísel. V každém cyklu je vygenerovaná náhodná hodnota (pomocí funkce uniform u které mají všechna možná čísla stejnou pravděpodobnost výskytu na rozdíl od RN) která nabývá hodnot od 0 do 10^6. V případě že hodnota četnosti čísla v poli, na které ukazuje výše zmíněný čítač *counter_sig*, je větší, než číslo vygenerované pomocí uniform nastane vygenerování poruchy. Porucha je vytvořena tak, že opět pomocí funkce uniform je vytvořeno celé číslo o velikosti od 0 do 23. Tato hodnota se použije jako index pro 24bitový signál *f_description* a takto vybraný bit tohoto signálu je nastaven do log. 1. Tento 24bitový signál slouží jako vstup do jednotlivých bloků FI (kterých je v tomto případě rovných 24), tedy každý bit přísluší jinému bloku FI a skrze něj se přenáší informace o tom na kterém bloku se má vyskytnou porucha.

Při vygenerování poruchy se zapíšou do textového souboru s názvem faults_report.txt informace o této poruše. Konkrétně tedy kolikátá porucha nastala, na kterém FI se porucha projeví a čas jejího vzniku. Signál f_description nese pouze informaci o vzniku poruchy a jejím výskytu, délku poruchy řeší až bloky FI. Z výše napsaného popisu generace poruchy je zřejmé že v jediný okamžik může nastat

přechodová porucha pouze na jednom místě (to neznamená, že v jeden okamžik nemůže být více přechodových poruch zaráz)

FD má čtyři volitelné parametry které ovlivňují rychlost změny indexu pole, tedy čítače *counter_sig* a pravděpodobnost vzniku poruchy. Prvním parametrem je *time_delay* který udává maximální hodnotu druhého čítače (*count_sig*) který při dosažení maximální hodnoty inkrementuje o jedna hodnotu hlavního čítače (*counter_sig*). Tedy hodnota parametru *time_delay* určuje o kolik hodinových cyklu se prodlouží využití jedné hodnoty četnosti uložené v poli (tedy pokud je jeho hodnota rovna 0 tak se čítač *counter_sig* inkrementuje s každým hodinovým cyklem). Druhým parametrem je *time_speed* který vyjadřuje číslo o které se *counter_sig* inkrementuje. Tedy jeho zvýšením se začnou přeskakovat následující hodnoty četností uložené v poli (pokud je jeho hodnota rovna dvěma tak se každá druhá hodnota vynechá. Posledními dvěma parametry jsou *chance_multi* a *chance_divi* jejichž poměrem se násobí maximální hodnota při výpočtu pravděpodobnosti vzniku poruchy. Pomocí těchto dvou parametrů se tedy může pravděpodobnost vzniku poruchy snížit i zvýšit.

Jak je z popisu patrné FD hodnoty získané od RN přemění, tak, že hodnoty čísla změní na čas podle parametrů *time_delay* a *time_speed* a četnost vygenerovaných čísel na pravděpodobnost vygenerování poruchy. Při změně časové osy pomocí těchto parametrů je možné zkorigovat i osu pravděpodobnostní pomocí parametrů *chance_multi* a *chance_divi* tak aby poměr mezi oběma osami zůstal nezměněn. Díky tomu může být RN nahrazen jiným (principiálně shodným) generátorem s tím, že jeho výstupní hodnoty budou pomocí těchto čtyř parametrů zkorigovány podle potřeby. Níže je uveden příklad možného výskytu poruch v čase při parametrech *time_delay* = 49, *time_speed* = 2, *chance_multi* = 8.0 a *chance_divi* = 2.0.



Obr. 17: Příklad vzniklých poruch v čase

3.1.3 Blok Fault Injector (FI)

Z popisu FD se může zdát, že injektování poruch má na starosti právě blok FD. Ve skutečnosti tomu tak není, neboť se sítí směrovačů jsou, ze všech vyjmenovaných bloků zabývajících se poruchami, přímo propojeny pouze bloky FI, které do systému vnášejí jak trvalé, tak i přechodné poruchy podle impulzů u FD. Kromě vnášení poruch do systému FI také slouží jako sledovač procházejících signálů.

FI obsahuje čtyři parametry, z toho jsou tři nosiči názvů (dva číselné a třetí textový) a jeden co ovlivňuje funkci FI (*f_mod*). 1bitový parametr *f_mod* určuje v kterém ze dvou módů bude FI pracovat. Pokud je tento parametr roven log. 1, pak FI pracuje jako sledovač procházejících dat s možností vzniku přechodné poruchy. Pokud je tento parametr roven log. 0, jsou oba výstupy uzemněny (nastaveny na log. 0) a nezáleží na změně žádného ze vstupních signálů. Další parametry nesou informace o číslech směrovačů, mezi kterými je FI zapojen a číslo samotného FI.

V případě normálního chodu, tedy *f_mod* = 1, FI pouze propojuje dva směrovače a do textového souboru "sledovani_dat/FIxx" zapisuje zahájení přenosu dat a ukončení přesunu dat. Do textového souboru také kvůli větší přehlednosti vpisuji i směr toku dat, ukázka textu je zobrazena níže (obr. 18).

```
Data jsou posilana ze smeru 0 na 1 v case: 1795000 ps
Data z 0 na 1 odeslana v case: 1905000 ps
Data jsou posilana ze smeru 0 na 1 v case: 1955000 ps
Data z 0 na 1 odeslana v case: 2065000 ps
```

Obr. 18: Ukázka hlášení FI o průchodu dat.

Protože data mezi směrovači se posílají oběma směry, je nutné, aby FI obsahovala dvakrát ty stejné datové propojení pouze s opačnými vstupy/výstupy. Stejně tak musí obsahovat i dva sledovací systémy, neboť data mohou cestovat oběma směry zároveň, jak je možné vidět níže (obr 19).

```
Data jsou posilana ze smeru 1 na 0 v case: 6835000 ps
Data jsou posilana ze smeru 0 na 1 v case: 6855000 ps
Data z 1 na 0 odeslana v case: 6945000 ps
Data z 0 na 1 odeslana v case: 6965000 ps
```

Obr. 19: Ukázka hlášení FI o protichůdném průchodu dat v jednu chvíli

Jediné, co je v FI unikátní je projev poruchy, ta se promítne shodně na obou datových linkách. Injektování poruchy nastane po přijetí signálu z FD skrz 1bitový vstupní signál (*f_pulse*) který je nastaven do log. 1 na jeden hodinový cyklus. Po přijetí tohoto impulzu se nastaví časovač, který se s každým dalším hodinovým cyklem dekrementuje o jedna a při jeho vynulování se FI vrátí do původního stavu v jakém byl před poruchou, návrat do původního stavu trvá 50 µs.

FI obsahuje dvě funkce uniform které vytvářejí se signálem clk vytváření dva náhodné vektory, 15bitový a 14bitový, které se po přijetí poruchového impulzu složí a vloží společně s "000" jako maska pro výstupní signál. První tři bity flitů jsou tak pomocí třech nul od poruchového stavu ušetřeny a z toho důvodu je maximální množství ovlivnitelných bitů rovno 29.

Výstupní signály jsou tedy rovny výsledkem mezi vstupním signálem a logickou operací or s vytvořenou 32bitovou maskou. Tedy v případě přechodné poruchy je velice malá nenulová pravděpodobnost, že bity ovlivněné poruchou svou hodnotu nezmění (neboť již jsou v log. 1). Tuto pravděpodobnost ale snižuje fakt, že přechodná porucha trvá 50 µs což je několikanásobně větší čas, než je potřeba pro přenos dat přes blok FI (tyto časy se pohybují o dva řády níže (obr. 18 a 19)).

3.2 Testovací rozhraní

Testovací rozhraní zde nahrazuje NI a PE, a to jak v ohledu zpřístupnění komunikace přes lokální vstupy/výstupy směrovačů tak i ve vytváření paketů a jejich následovná distribuce do sítě směrovačů. Testovací rozhraní je z velké části využito beze změn z původního projektu. [1]

3.2.1 Nahrazení síťového rozhraní (NI)

Jak bylo již v kapitole (2.4) popsáno síťové rozhraní pracuje jako prostředník mezi směrovačem a výkonnou jednotkou a zajišťuje přeformátování dat tak aby je jednotlivé části mohli zpracovat. V tomto případě, kde jde čistě o testování sítě směrovačů je využití výkonných jednotek zbytečné a složité. Z toho důvodu tedy není potřeba aby data přicházející a opouštějící síť směrovačů byly přetransformovány, tedy není potřeba síťového rozhraní jako takového. Stále je ale nutné odněkud data brát a posílat, aby mohlo dojít k otestování sítě. [1] [2]

Toto řeší testovací rozhraní, které nahrazuje síťové rozhraní, a to takovým způsobem, že všechny lokální vstupy a výstupy směrovačů jsou k němu připojeny.

Pokud směrovač posílá nějaké data na svůj lokální výstup testovací rozhraní automaticky uděluje směrovači informaci, pomocí procedury *get_packet*, že je místo v paměti síťového rozhraní (ke kterému ovšem směrovač není připojen) a může data tímto směrem poslat dále. Tato část data od směrovače neukládá, ale pouze z nich zjistí potřebné informace jako místo původu, cílová lokalita (které se nacházejí na hlavičkovém flitu) a poté tento flit nahrazen následujícím tedy prvním středovým flitem z kterého jsou vyčteny informace o velikosti paketu a jeho identifikační číslo. Vždy při přijetí nového flitu (jakéhokoliv) se inkrementuje hodnota čítače o jedna. [1] [2]

Poté testovací rozhraní další flity čistě přemazává následujícími flity, a to až do

doby kdy dorazí koncový flit. Po příchodu koncového flitu porovná hodnotu čítače s informací ohledně délky paketu, pokud tyto hodnoty nesouhlasí je vytvořena varovná zpráva, stejně tak zkontroluje cílovou adresu se svojí a při neshodě je také vytvořena varovná zpráva. Pokud není rozdíl v informací je vytvořena zprava která je uložena do textového souboru "received.txt" kde jsou vypsány veškeré informace (obr. 20).[1],[2]

```
Packet received at 3095000 ps From: 3 to: 2 length: 8 actual length: 8 id: 6
Packet received at 3185000 ps From: 10 to: 2 length: 8 actual length: 8 id: 5
```

Obr. 20: Ukázka hlášení úspěšně přijatého paketu [1] [2]

Aby případně vytvořené pakety mohli proniknout do sítě směrovačů tak je potřeba využít proceduru *credit_counter* která je čistě 2bitový čítač stejný, jak je popsán v kapitolách (2.3.4 a 2.4) jehož funkcí je dodržovat vhodný poměr mezi volným místem v lokální paměti směrovače a existencí flitů které je třeba tímto směrem poslat.

Testovací rozhraní taktéž obsahuje proceduru určenou k vytvoření paketů (*get_random_packet*). Pro správnou funkčnost je třeba proceduře vložit správné informace které jsou velikost sítě v obou osách (tedy x a y), počáteční adresa, zpoždění, minimální a maximální rozměry, čas pro zakončení generování dalších paketů, důležité pro správnou funkčnost je propojení s procedurou *credit_counter*. Počáteční adresa se využije jako základ pro generování pseudonáhodného čísla, tím se zajistí rozdílnost mezi pakety z různých počátečních adres. S využitím náhodné hodnoty se vytvoří hodnota délky paketu a pokud tato hodnota splní velikostní limity zůstane nepozměněna, v opačném případě je změněna na bližší limitní hodnotu. Podobným způsobem je vypočítána hodnota mezi výpočtu, zpoždění mezi dalším generováním, jehož výsledek se dostane po přičtení délky paketu. Dále je pomocí rozměrů sítě a náhodného čísla vypočítána cílová adresa. Pomocí těchto informací je vytvořena zpráva, která je uložena do textového souboru s názvem "send.txt" a až poté se začnou generovat flity počínaje hlavičkovým. Do flitů nesoucích datové informace jsou jako data vloženy vygenerovaná náhodná čísla. Vytvoření flitu může začít vždy až po odeslání předchozího, protože testovací rozhraní nedisponuje pamětí, kde by mohli setrvávat, to znamená že celý paket může být vytvořen pouze za předpokladu, že mu nic nebrání v průchodu směrem k cílové destinaci. Po vytvoření koncového flitu se spustí čítač, který zpozdí vytvoření dalšího paketu, který již bude mít jiné hodnoty, neboť náhodná hodnota se již během první generace stihla několikrát změnit. Generování paketů probíhá tak dlouho, než nenastane čas zadaný jako parametr procedury, po jeho dosažení se dokončí generace posledního (již rozdělaného) paketu. [1][2]

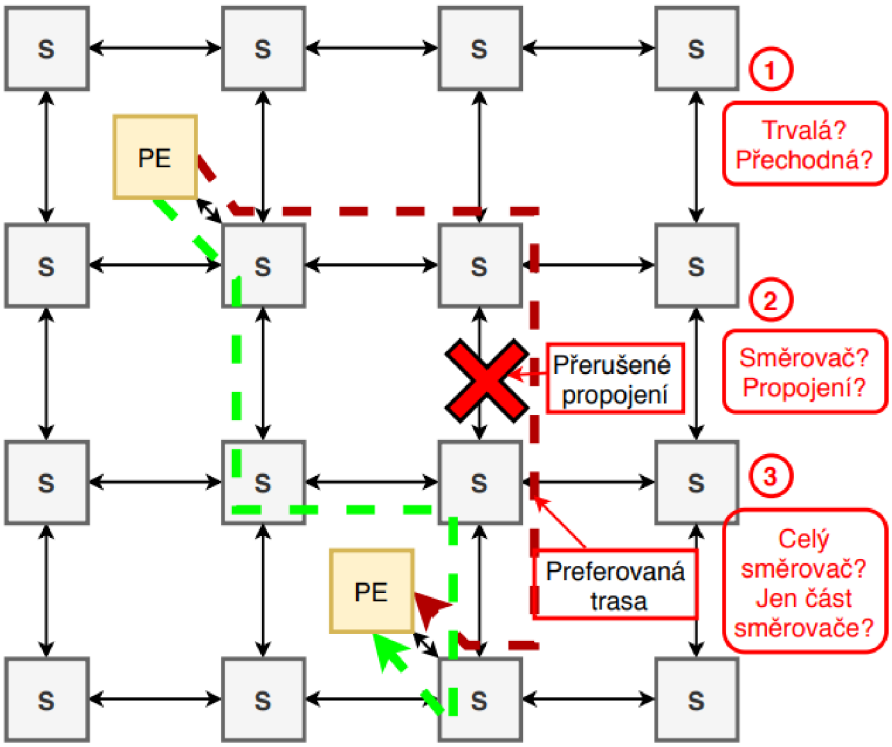
4 DIAGNOSTIKA

V této kapitole bude rozebrána metoda přesné lokalizace a identifikace poruchy (PLIP), její výsledky a vliv na velikost sítě a její spolehlivost. Z důvodu kompatibility a faktu, že je použita síť o velikosti 4×4 (tedy neobsahuje velké adresační souřadnice), je celkový systém sítě upraven tak aby používal pouze 16bitové flity, tedy datová informace je snížena o 16 bitů (tedy na 12 bitů) a v případě adres, ID a informace o délce paketu je velikost snížena o 8 bitů (tedy na 6 bitů).

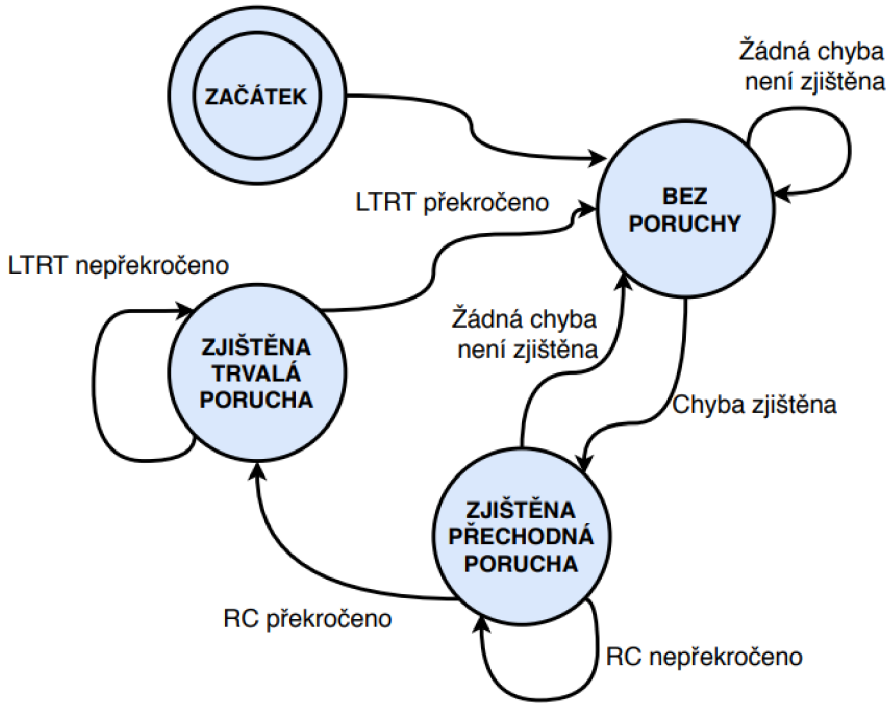
4.1 Přesná lokalizace a identifikace poruchy (PLIP)

Metoda přesné lokalizace a identifikace poruchy (PLIP) se zaměřuje na propojovací komunikační síť NoC s cílem identifikace trvalých a přechodných poruch, rozlišit mezi poruchou směrovače a datové linky mezi směrovači. Dále rozpoznat poruchy na jednotlivých datových cestách uvnitř směrovače. [6]

K rozlišení mezi poruchami trvalými a přechodnými se používají dva koncepty: „retransmission credit“ (RC), přiřazený každé propojovací cestě, a „long transient recovery timeout“ (LTRT), umožňující jednou označenou linku/cestu jako trvale porouchanou přezkoumat, není-li jen přechodnou poruchou dlouhou. [6]

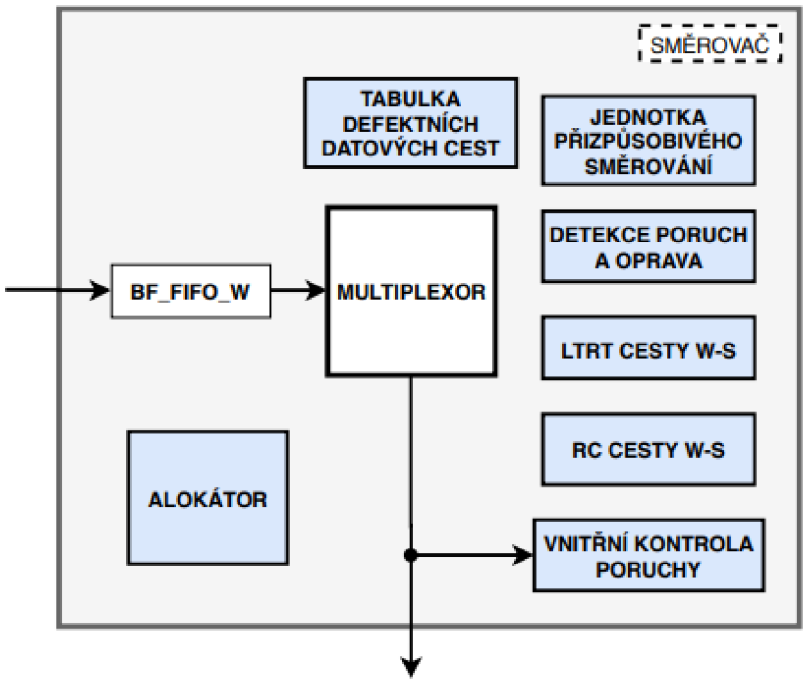


Obr. 21: Cíle přesné lokalizace a identifikace [6]

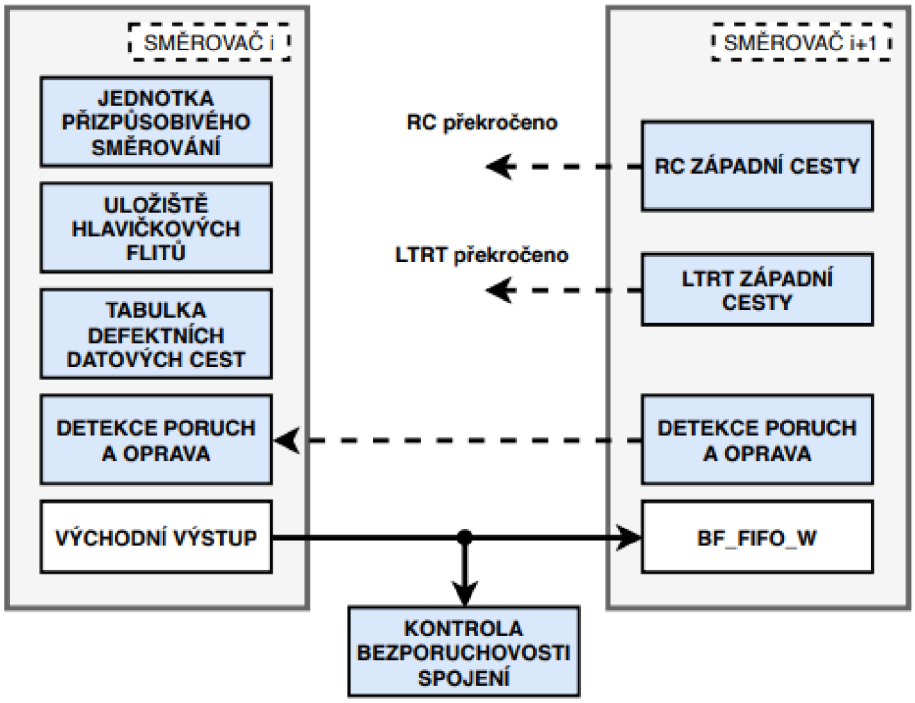


Obr. 22: Základní algoritmus identifikace trvalé a přechodné poruchy [6]

PLIP ke všem komponentám, na nichž má být provedena lokalizace poruchy (2, 3), používá komponenty z obr. 21 – na datových linkách mezi směrovači a na datových cestách uvnitř směrovačů. [6]



Obr. 23: Lokalizace poruchy na datových linkách uvnitř směrovače [6]



Obr. 24: Lokalizace poruchy na datových linkách mezi směrovači [6]

4.2 Experimentální výsledky

Pro zjištění efektivnosti užití metody PLIP, zajišťující bezporuchovost komunikační sítě v NoC, byly ve Windows 7 provedeny dvě sady experimentů. První sada (tab. 3) vedla k určení výkonnostních a spolehlivostních parametrů sítě užitím simulátoru ModelSim SE 6.1f – tj. latence paketů, propustnost sítě a střední doba mezi poruchami (MTBF).

Tab. 3: Výkonnostní a spolehlivostní parametry v přítomnosti žádné a jedné trvalé poruchy

Injekce poruch [%]	Latence paketů [sim_clks]		Propustnost sítě [%]		Střední doba mezi poruchami (MTBF) [sim_clks]	
	0 trvalých	1 trvalá	0 trvalých	1 trvalá	0 trvalých	1 trvalá
0	21,6	21,8	100	100	x	x
0,12	21,8	22,1	98,8	98,6	1818	1538
0,32	22,5	22,9	95,9	95,2	526	455
0,92	23,6	24,7	91,3	88,4	247	189

Není-li v tabulkách uvedeno jinak, byly experimenty provedeny s následujícími parametry: mřížková propojovací síť velikosti 4x4, 20.000 simulačních cyklů, 8flitové

pakety, 16bitové flity, rychlost generování paketů do NoC průměrně 0,0625 paketů na simulační cyklus, zabezpečení komunikace paritním bitem, „retransmission credit“ RC velikosti 3, „long transient recovery timeout“ (LTRT) velikosti 15.

Druhá sada experimentů (tab. 4) vedla k určení množství zabraných prostředků FPGA v počtu LUT (look-up tables) a klopných obvodů typu D (FF) užitím Xilinx ISE 14.7. PLIP představuje přijatelné navýšení počtu zabraných LUT na mřížkové sítí velikosti 4x4 (přibližně 25 %), zatímco v počtu FF je navýšení poněkud výraznější (přibližně 40 %). Počet FF je možné ovlivnit nastavením statistických parametrů designu RC a LTRT, které mají vliv na rozlišení trvalé, přechodné a dlouhé přechodné poruchy a které je možné přizpůsobit „kvalitě“ prostředí, v němž má být systém NoC provozován.

Tab. 4: Množství zabraných prostředků FPGA v počtu LUT a klopných obvodů

Cílové FPGA Xilinx	Bonfire 4x4 NoC bez PLIP		Bonfire 4x4 NoC s PLIP	
	#FF	#LUT	#FF	#LUT
Spartan-6 XC6SLX16	1.184	403	1.640	512
Spartan-3 XC3S2000	1.184	428	1.640	546

ZÁVĚR

Úkolem této práce bylo použití číslicového modelu architektury NoC pro diagnostické účely a pro zvyšování výkonosti. Nejprve bylo potřeba vybrat vhodný model a naučit se a popsat jeho funkci. Jako model byla vybrána NoC architektura směrovače vytvořená v rámci projektu nazvaném Bonfire. Jedná se o směrovač určený pro zapojení do mřížky. Vyrovnávací paměti jsou typu FIFO a zvládnou pojmout maximálně 4 flity, které jsou 32bitové a pro tuto práci byli upraveny na 16bitové, ovšem kvůli způsobu řízení toku dat lze vždy najednou uložit pouze 3 flity. Jelikož směrovač využívá princip přeposílání dat červí dírou není malá velikost vyrovnávacích pamětí vůbec důležitá.

Pro tento model byly vytvořeny nesyntetizovatelné, tedy slouží čistě k použití v simulačním prostředí, bloky jejichž úkolem je vnášet poruchovost okolního prostředí do architektury. Jako základ pro vytváření poruch byla použita charakteristika blížící se tvarem Gaussově rozdělení, která byla vytvořena z 10^6 pseudonáhodných čísel vytvořených blokem RN. Jelikož vygenerování takového množství čísel vyžaduje simulaci trvající v případě použití pouze bloku RN až v desítkách minut. Pokud by tedy tyto čísla mely být generovány při každé simulaci celé architektury (která je mnohem složitější než samotný blok RN) výrazně by se zvýšil čas takové simulace (až o několik desítek minut) a pokaždé by byly vygenerovány stejné výsledky (jelikož vygenerovaná čísla vycházejí ze čtyř základních 32bitových hodnot a dokud nedojde ke změně alespoň jedné z nich budou výsledky vždy totožné). Z těchto důvodů jsou výsledky uloženy v textovém souboru, ze kterého jsou poté použity dále.

Pro zpracování těchto dat byl vytvořen blok FD, který pomocí svých čtyřech proměnných dokáže z četností jednotlivých vygenerovaných čísel generovat pseudonáhodné poruchy které dále distribuuje do sítě. V zásadě je tento blok vytvořen tak, že může vstupní hodnoty (v tomto případě Gaussovo rozdělení) zpracovávat tak aby doba trvá této funkce byla nastavitelná, a to tak že u každé hodnoty se může zastavit na více hodinových cyklů (čas trvání funkce se zvyšuje stejně jako pravděpodobnost nastání poruchy) a/nebo může zvolený počet následujících hodnot přeskočit (doba trvání funkce a pravděpodobnost výskytu poruchy se zmenší). Neboť se tímto mění i pravděpodobnost výskytu poruchy může se tato změna kompenzovat prostřednictvím dvou proměnných na jejichž poměru tato pravděpodobnost přímo závisí. Tedy tento blok dokáže zpracovat i velikostně jiné funkce.

Pro konečnou injektáž poruchy do architektury byly vytvořeny bloky FI, které spojují vždy dva sousední směrovače jejichž vzájemné vstupy/výstupy může trvale uzemnit (do log. 0) nebo po přijetí impulzu od FD maskuje výstupy přes pseudonáhodně vytvořenou masku (mezi maskovacím vektorem a výstupem je

proveden logický or). Také vytváří zprávy o pocházejících datech (s časem jejich průchodu).

S Využitím těchto bloků a testovacího rozhraní které se stará o generování a příjímání paketů do a ze sítě byla provedena úprava směrovačů tak aby dokázaly přesně identifikovat a analyzovat poruchy (PLIP) jak mezi směrovači, tak i přímo vně směrovači (takovouto poruchu systém generování poruch nedokáže vytvořit) a za pomocí těchto poznatků zvýšit výkonnost sítě. Z toho důvodu museli být upraveny flity na velikost 16 bitů z původních 32 bitů.

Výsledky experimentů, které byly prezentovány na studentské soutěži EEICT 2020 [8], je možné shrnout do několika zjištění o efektivnosti užití metody PLIP (kap. 4.1) pro zajištění bezporuchovosti komunikační sítě v NoC. PLIP zaručuje jen nepatrně horší výkonnost a spolehlivost NoC v případě, kdy je v NoC přítomna jedna trvalá porucha s občasným výskytem poruch přechodných, oproti případu, kdy není přítomna žádná trvalá porucha s občasným výskytem poruch přechodných. Na druhou stranu dosažení velmi příznivých výkonnostních a spolehlivostních parametrů je vyváženo nezanedbatelnou spotřebou zabraných prostředků FPGA (LUT a klopných obvodů). PLIP je tedy velmi efektivní implementovat v prostředích se zvýšeným výskytem rušení, např. radiačním či elektromagnetickým, nebo v systémech vyžadujících vysokou spolehlivost, např. v systémech veřejné dopravy. Naopak implementace v „nekritických“ systémech je poměrně neefektivní.

LITERATURA

[1] TALLINN UNIVERSITY. Bonfire Project, rev. 9 [VHDL design]. 4. dubna 2019 [přístup 13. října 2019]. Dostupný z: https://github.com/Project-Bonfire/Bonfire

[2] AZAD, S. P., NIAZMAND, B., JANSON, K., et al.: From Online Fault Detection to Fault Management in Network-on-Chips: A Ground-Up Approach. In: *Proceedings – IEEE Int. Symp. on Design and Diagnostics of El. Circuits & Systems (DDECS)*. New York: IEEE, 2017, s. 48–53. ISBN 978-1-5386-0472-4

[3] WEN-CHUNG, T., YING-CHERNG, L., SAO-JIE, Ch. *Networks on Chips: Structure and Design Methodologies* [online]. Hindawi Publishing Corp., 2011 [cit. 2019-12-06]. Dostupné z: https://www.researchgate.net/publication/220588483_Networks_on_Chips_Structure_and_Design_Methodologies

[4] THAKYAL, Deewakar a Pushpita CHATTERJEE. *Dia-Torus: A Novel Topology For Network On Chip Design* [online]. Internacional Journal of Computer Networks & Communications. 8(3), 2016 [cit. 2019-12-07]. s 137-148. DOI 10.5121/ijcnc.2016.8310. Dostupné z: https://www.researchgate.net/publication/303870995_Dia-torus_A_novel_topology_for_network_on_chip_design

[5] DIMITRAKOPOULOS, Giorgos, Anastasios PSARRAS a Ioannis SEITANIDIS. *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective* [online]. New York: Springer, 2015 [cit. 2019-12-22]. ISBN ISBN 978-1-4614-4301-8. Dostupné z: http://s1.nonlinear.ir/epublish/book/Microarchitecture_of_Network_on_Chip_Routers_9781461443001.PDF

[6] ŠŤÁVA, M. Efficient Error Recovery Scheme in Fault-tolerant NoC *Architectures*. In: *IEEE Int. Symp. on Design and Diagnostics of El. Circuits & Systems (DDECS)*, 2019. ISBN 978-1-7281-0073-9

[7] FORSTÉN, H. Generování normálně distribuovaných pseudonáhodných čísel. *Henrik's Blog* [online]. 14. dubna 2020, 1:39 [vid. 2012-10-23]. Dostupné z: https://hforsten.com/generating-normally-distributed-pseudorandom-numbers-on-a-fpga.html

[8] VALACHOVIČ, M. On the Efficiency of Precise Fault Localization and Identification in NoC. In: *Proceedings of the 26th Conference STUDENT EEICT 2020*. Brno: FEKT VUT, 2020. (*V tisku*)

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

NoC	Network on Chip
NI	Network Interface, síťové rozhraní
PE	Processing Element, výkonná jednotka
MUX	Multiplexer, multiplexor
XBAR	Crossbar Switch
LBDR	Logic-Based Distributed Routing
PLIP	Přesná lokalizace a identifikace poruchy
RN	Random_number
FI	Fault Injector
FD	Fault Description

SEZNAM OBRÁZKŮ

SEZNAM TABULEK