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

IMPLEMENTACE KONCOVÉHO BODU V SÍTI SPACEWIRE DO FPGA
SPACEWIRE ENDPOINT IMPLEMENTATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Marek Hráček
AUTHOR

VEDOUCÍ PRÁCE — Ing. Vojtěch Dvořák
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Mikroelektronika a technologie**

Ústav mikroelektroniky

Student: Marek Hráček ***ID:*** 164288

Ročník: 3 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Implementace koncového bodu v síti SpaceWire do FPGA

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte specifikaci sítě SpaceWire a navrhněte modul fungující jako koncové zařízení v této síti. Modul bude umět přijímat i odesílat datové pakety, zpracovávat časové kódy a na základě požadavku nadřazeného systému iniciovat komunikaci s dalším koncovým bodem v síti. Modul popište v jazyce VHDL a implementujte do zvoleného obvodu FPGA.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 8.2.2016 ***Termín odevzdání:*** 2.6.2016

Vedoucí práce: Ing. Vojtěch Dvořák
Konzultant bakalářské práce:

doc. Ing. Jiří Háze, Ph.D., *předseda oborové rady*

Abstrakt:

Předkládaná práce se zabývá návrhem koncového rozhraní standardu SpaceWire, používaného vesmírnými plavidly pro komunikaci mezi palubními zařízeními a systémy. V teoretické části je popsán standard samotný, způsob provozu a jednotlivé logické vrstvy zastřešující různé funkce. V další části je rozebrán návrh koncového bodu, jeho jednotlivé bloky, představena jsou řešení jednotlivých funkcí. Nakonec je předloženo využití zdrojů po syntéze a dosažitelná rychlost v konkrétním FPGA.

Klíčová slova:

SpaceWire, komunikační síť, rozhraní, koncový bod, FPGA, VHDL

Abstract:

This work deals with the SpaceWire standard, that is used to convey the communication between modules and subsystems on board spacecrafts. Theoretical part describes standard, the way it operates and logic layers in which various functions are divided. Next part is describing design of SpaceWire endpoint itself. Presented are individual components and solutions to implement features of standard. Last chapter deals with device utilization and reached speed after synthesis with specific FPGA.

Keywords:

SpaceWire, communication network, interface, endpoint, FPGA, VHDL

HRÁČEK, M. *Implementace koncového bodu v síti SpaceWire do FPGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 43 s. Vedoucí bakalářské práce Ing. Vojtěch Dvořák.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma **„Implementace koncového bodu v síti SpaceWire do FPGA“** jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne …… .....
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Vojtěchu Dvořákovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne …… .....
podpis autora

Obsah

Seznam obrázků a tabulek

Seznam zkratek

AMBA	Pokročilá architektura mikrokontrolérových sběrnic (Advanced Microcontroller Bus Architecture)
CCSDS	Poradní komise pro kosmické datové systémy (Consultative Comittee for Space Data Systems)
DDR	Způsob operace digitálních obvodů reagující na náběžnou i sestupnou hranu hodinového signálu (double data rate)
ECSS	Evropská kooperace pro kosmickou standardizaci (European Cooperation for Space Standardization)
ESA	Evropská kosmická agentura (European Space Agency)
FIFO	paměť ve formě fronty, kdy data přijatá první, jsou první použita (First In, First Out)
FPGA	programovatelné hradlové pole (Field Programmable Gate Array)
IEEE	Institut pro elektronické a elektronické inženýrství (Institute of Electrical and Electronics Engineers)
IP	duševní vlastnictví (Intellectual property)
NASA	Národní úřad pro letectví a kosmonautiku (National Aeronautics and Space Administration)
RAM	paměť s přímým přístupem (random-access memory)
RMAP	Protokol pro vzdálený přístup do paměti (Remote Memory Access Protocol)
TIA/EIA	Asociace telekomunikačního průmyslu/Aliance elektronického průmyslu (Telecommunications Industry Association/Electronic Industries Alliance)
UCF	soubor s omezeními pro implementaci návrhu (user constraint file)
VHDL	popisný jazyk určený k tvorbě digitálních obvodů (VHSIC Hardware Description Language)
VHSIC	velmi rychlé integrované obvody (Very High Speed Integrated Circuits)

Úvod

SpaceWire je komunikační síť, používaná ve vesmírných plavidlech, která propojuje různé nástroje, senzory, paměti, procesory a další palubní systémy. Je koncipována jako jednoduchá k implementaci, dosahuje vysokých rychlostí, nízké spotřeby a je velmi flexibilní. Podoba sítě může sahat od dvou spojených bodů po komplexní síť, obsahující směrovací přepínače. SpaceWire spoje jsou sériové, obousměrné a plně duplexní [1][2].

SpaceWire je veden organizací ECSS jako standard ECSS-E-ST-50-12C. Publikován byl v roce 2003, jeho autorem je Steve Parkes ze skotské Univerzity v Dundee. Dále se podílelo mnoho lidí z ESA, NASA a dalších organizací. SpaceWire vychází ze standardu IEEE 1355-1995, jehož vývoj začal v roce 1992. V té době byla nutnost po systémech digitálního zpracování signálu, které nebylo možné implementovat v jednom čipu. Tento standard tedy měl zajistit komunikaci mezi paralelně pracujícími výpočetními prvky. I přesto, že byl použit na několika misích, obsahoval mnoho problémů a nedostatků při použití ve vesmírných aplikacích. ESA proto zadala Univerzitě v Dundee, aby je vyřešila, z čehož vznikl standard SpaceWire. Spravován je společností STAR-Dundee Ltd, kterou založil Steve Parkes a je těsně navázána na činnost Univerzity v Dundee [1][3].

Spolu se SpaceWire mohou být použity další protokoly, což rozšiřuje možnosti aplikace. Aby si nepřekážely a mohly procházet sítí bez problémů, je definována identifikace protokolů (ECSS-E-ST-50-51C), která umožňuje rozeznat protokol použitý v konkrétním paketu, případně jej zahodit, pokud jej nezná. Mezi protokoly, které jsou často používané, patří RMAP (ECSS-E-ST-50-52C), pomocí kterého lze přistupovat k pamětím ve vzdálených zařízeních přes síť SpaceWire. Další je CCSDS Packet Transfer Protocol (ECSS-E-ST-50-53C), který se stará o přenos CCSDS paketů skrz síť SpaceWire tak, že je vloží do SpaceWire paketu [1][4].

Podle standardu byly poté vytvořeny různé implementace jak koncových bodů, tak přepínačů. Lze je zakoupit jako zdrojové kódy k použití na vlastních projektech. Jádra nabízí ESA a společnost Aeroflex Gaisler.

ESA nabízí jádro SpaceWire Codec [5], základní rozhraní, spravující tok dat po spoji, přičemž dovoluje nakonfigurovat některé parametry. Dále jsou nabízena rozšíření, která realizují protokol RMAP [6] nebo sběrnici AMBA [7], jež slouží k vnitročipové komunikaci [8].

Aeroflex Gaisler nabízí jádra GRSPW a GRSPW2 [9], což jsou SpaceWire rozhraní rozšířená o protokol RMAP a identifikaci protokolu. Také obsahují sběrnici AMBA. Rovněž je možné zakoupit jádro GRSPW_CODEC [10], které tato rozšíření neobsahuje a pro komunikaci s obsluhovaným zařízením využívá 9 bitů širokou FIFO

s konfigurovatelnou délkou. Všechna jádra mohou také obsahovat ochranu na RAM pamětích proti SEU (single event upset), což je dočasná chyba, způsobující změnu stavu v logickém obvodu důsledkem zásahu ionizující částicí.

Tyto jádra jsou ale cenově velmi nevýhodná, náklady na pořízení se pohybují v řádech statisíců korun. Proto cílem práce je vyvinout vlastní návrh rozhraní koncového bodu, jež bude ovládat všechny základní funkce standardu, a které bude možné v budoucnu modifikovat pro podporu další funkcionality. Výhodou je, že SpaceWire byl koncipován jako jednoduchý k implementaci, což usnadňuje nasazení a zvyšuje spolehlivost. Výsledný design by tak měl zabírat 5000-8000 logických hradel [11].

1 Popis standardu

Standard je rozdělený na několik vrstev, které popisují různé části sítě, od konektorů až po abstraktnější věci, určující podobu komunikace, strukturu sítě, atd. [2].

1. Fyzická vrstva – kabely, konektory, desky plošných spojů
2. Signálová vrstva – elektrické charakteristiky, kódování a časování signálů
3. Znaková vrstva – enkódování datových a ovládacích znaků
4. Výměnná vrstva – způsob operace samotného spoje, inicializace komunikace, běžný provoz a detekce a odstranění chyb
5. Paketová vrstva – struktura paketů obsahující data
6. Síťová vrstva – struktura a provoz sítě

1.1 Fyzická vrstva

Popisuje mechanické a elektrické parametry kabelů, konektorů a desek plošných spojů tak, aby byly splněny požadavky pro elektromagnetickou kompatibilitu ve vesmírných plavidlech, a také to, že sítě SpaceWire komunikují ve velkých rychlostech [1].

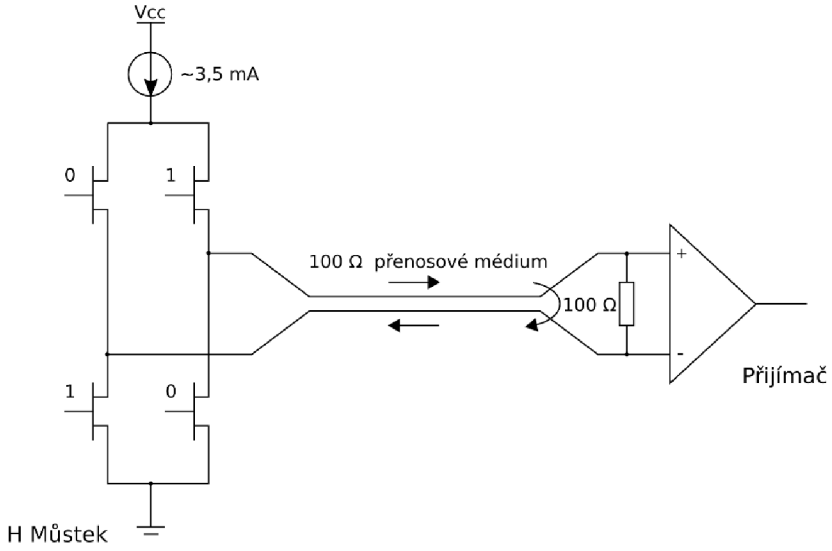
1.2 Signálová vrstva

1.2.1 Signálové úrovně a šumové hranice

SpaceWire používá na přenos signálu LVDS (Low voltage differential signalling), což je standard známý také jako TIA/EIA-644 [12]. Sestává ze dvou vodičů s rozdílovou impedancí 100 Ω, spojených na straně přijímače 100 Ω rezistorem, na němž se měří napětí, které má typickou hodnotu 350 mV. To je způsobeno proudovým zdrojem na straně vysílače. Orientace proudu na vodiči se nastavuje pomocí H můstku (viz **Obr. 1**).

Hranice pro přepnutí úrovně jsou přibližně ±100 mV od nulového rozdílového napětí, které je typicky 1,2 V posazené nad zemí. Tato hodnota se může díky šumu pohybovat ±1 V a spoj bude stále fungovat.

Výhodou této technologie je vysoká odolnost proti rušení. Protože proudy ve vodičích jsou orientovány opačně vůči sobě, indukovaný šum se navzájem vyruší, pokud budou vodiče zakroucené do sebe. Dále použití nízko-napěťových přechodů snižuje spotřebu – pohybuje se kolem 50 mW na řadič [1].



Obr. 1: Schéma LVDS spoje

1.2.2 Přenosová rychlost

Přenosová rychlost je standardem určená jako rozsah minimální a maximální rychlosti.

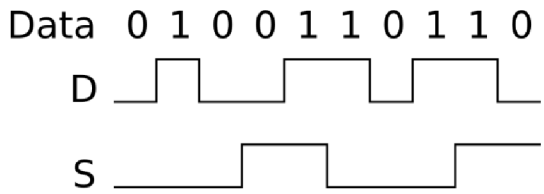
Minimální je odvozena od časové prodlevy při nečinnosti spoje, po které se resetuje spojení. Tato prodleva je 850 ns, což je rychlost 1,18 Mbit/s. Použitá přenosová rychlost tedy musí být vždy větší, a proto je minimální rychlost stanovena na 2 Mbit/s.

Maximální rychlost je taková, při které na spoji nedochází k přenosovým chybám kvůli skew (časová nesouhlasnost signálů) a jitteru (náhodně proměnná frekvence) signálu.

Pracovní přenosová rychlost pak může být nastavena kdekoliv v tomto rozsahu, přičemž rychlosti obou směrů se mohou lišit. Po resetu či odpojení po neaktivitě však oba směry vždy začnou vysílat při rychlosti 10 Mbit/s (±1 Mbit/s) a až po správném navázání komunikace mohou přejít na rychlost jinou [2].

1.2.3 Datové enkódování

SpaceWire používá enkódování Data-Strobe (DS). Při něm se příjemci neposílá s datovým signálem hodinový signál, ale tzv. Strobe signál, který mění svoji hodnotu vždy, když datový signál zůstává konstatní. Hodinový signál lze pak obnovit operací XOR signálů Data a Strobe.



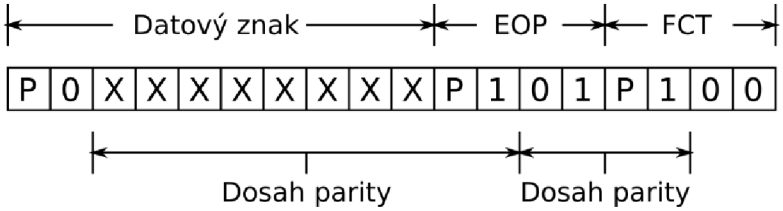
Obr. 2: DS enkódování

Mění se vždy pouze jeden signál, což zvyšuje toleranci nesouhlasnosti hodinového a datového signálu z půl bitu na 1 bit, na rozdíl od klasického přenosu signálu. Z principu však vyplývá, že hodinový signál obnovený XOR hradlem má poloviční frekvenci, než měl původní hodinový signál před použitím DS enkódování. Je proto třeba tomu přizpůsobit přijímač, nebo použít komplikovanější metody obnovy hodinového signálu [1].

1.3 Znaková vrstva

SpaceWire používá dva typy znaků ke komunikaci, datové a ovládací. Obsahují vždy 2 pomocné bity a 8 datových, respektive 2 ovládací bity.

První pomocný bit je lichý paritní bit, který se nastavuje tak, aby v pokryté oblasti byl lichý počet logických jedniček. Tato oblast sestává z předchozích 8 bitů datového znaku nebo 2 bitů znaku ovládacího, samotného paritního bitu a data-control bitu, který následuje. Tento druhý pomocný bit značí typ znaku, 0 znamená datový znak a 1 řídící znak [1].



Obr. 3: Dosah parity

V datových znacích pak po pomocných bitech následuje 8 bitů datových, kdy se posílá nejméně významný bit první.

V případě ovládacích znaků následují bity 2, jsou tedy 4 druhy těchto znaků [2]:

- 00 – FCT (Flow control token) slouží k inicializaci spoje nebo reguluje tok dat (jedno FCT značí 8 znaků)
- 01 – EOP (Normal end of packet) značí konec paketu
- 10 – EEP (Error end of packet) indikuje konec paketu, když dojde k chybě
- 11 – ESC (Escape) slouží k tvorbě ovládacích kódů

1.3.1 Ovládací kódy

Spojením ESC a FCT vznikne NULL kód, který slouží k tomu, aby nedošlo k odpojení spoje v důsledku nečinnosti.

Pokud je ESC následován datovým znakem, jde o časový kód, pomocí kterého se synchronizují zařízení v síti SpaceWire. Prvních 6 nejméně významných bitů nese informaci o čase, respektive jejich hodnota se používá k určení platnosti tohoto znaku. Další 2 bity jsou ovládací vlajky, které se v některých případech používají na rozlišení druhů časových kódů v síti, ale většinou jsou v logické 0 (viz 1.4.5).

Pokud je ESC následován ESC, EOP nebo EEP, dojde k chybě [2].

1.3.2 Znaky ve vztahu k vrstvám

Znaky a kódy se také dělí podle funkce na linkové, paketové a časové.

NULL a FCT se řadí mezi linkové znaky neboli L-znaky. Slouží k řízení komunikace na výměnné vrstvě a vyšší vrstvy je už nevidí. Datové znaky a značky konce paketů (EOP a EEP) se nazývají normální znaky (N-znaky) a tvoří obsah paketů. Časové znaky slouží k posílání času a synchronizaci informací a jsou předány časovému ovladači na příslušném koncovém bodu či směrovači.

SpaceWire rozhraní pak tyto 3 druhy prokládá jak je třeba. Pouze se nesmí smíchat N-znaky z jednoho paketu s druhým [1].

1.3.3 Chování vysílače po resetu

Po resetu jsou oba (Data a Strobe) signály nastavené do logické 0. První znak, který bude poté odeslán, bude mít paritní bit nulový, aby první změna úrovně proběhla na Strobe signálu [2].

1.4 Výměnná vrstva

Tato vrstva se stará o správný chod komunikace mezi dvěma konci spoje. Je v ní definována inicializace spoje, regulace toku, aby nedošlo k zahlcení přijímacích pamětí a také zvládání chyb a poruch.

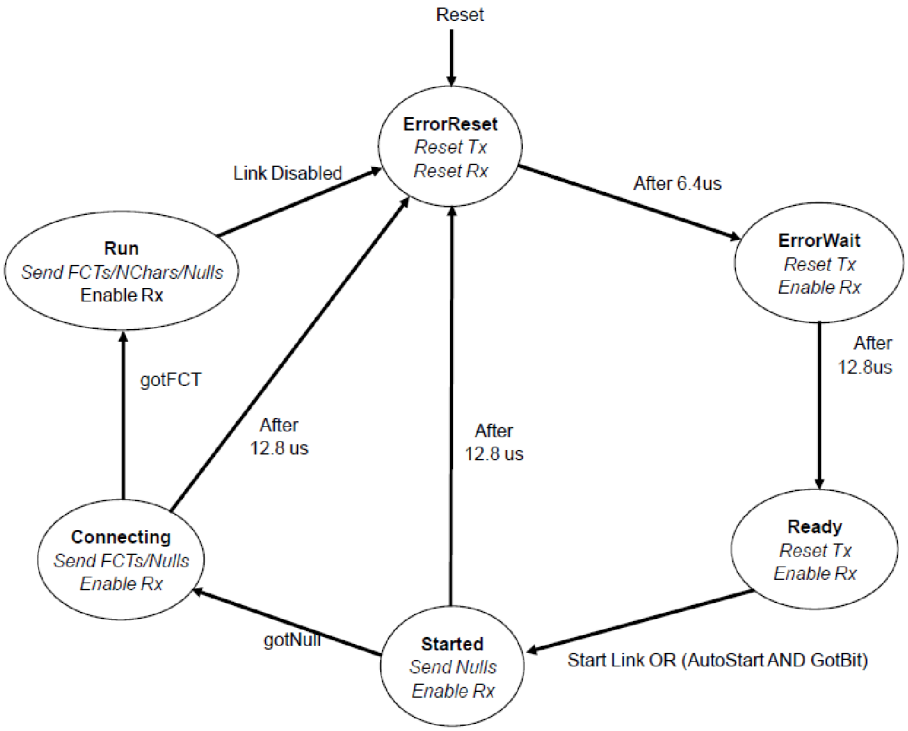
1.4.1 Inicializace spoje

Inicializaci je nutné provést, aby bylo zaručeno, že oba konce fungují správně, proto se to provádí tzv. podáním ruky (handshake). Také při tom dojde k bitové synchronizaci dekódováním data-strobových signálů k vytvoření datového a hodinového signálu. Pokud nedojde k úspěšné synchronizaci, provede se reset a proces se opakuje. Protože se

inicializace provádí pomocí kódu NULL, jehož sekvence nemusí být v datovém toku jedinečná, lze synchronizaci spolehlivě provést pouze při nyní.

Po resetu spoje je rozhraní určitou dobu vypnuto. Poté dojde k výměně znaků NULL/FCT. Každý konec posílá NULL znaky a čeká, až jej obdrží od druhého, poté začne posílat/čekat na FCT. Aby mohl přijít FCT, musel druhý konec úspěšně přijmout NULL a přijme tedy i FCT. Spojení tak bylo úspěšně navázáno [1].

Na **Obr. 4** je zjednodušený stavový diagram, který tento proces znázorňuje. Jsou zde definované prodlevy (6,4 µs a 12,8 µs), po kterých dojde k aktivaci nejprve přijímače a poté vysílače. Přechod označený Start Link pak značí pokyn k zapnutí spoje, který může přijít od zařízení, ke kterému konkrétní rozhraní patří, nebo může být způsoben tzv. Autostartem, viz 1.4.2.



Obr. 4: Stavový diagram průběhu funkce rozhraní [2]

Pokud jeden konec již vysílá NULL/FCT znaky, čeká na odpověď pouze stanovenou dobu (12,8 µs). Poté přejde zpátky do stavu *ErrorReset* a pokusí se projít celým procesem znovu.

Celý proces končí ve stavu *Run*, ve kterém jsou oba konce správně připojené a připravené posílat data. Ve stavu *Run* zůstanou, dokud jeden konec nedostane pokyn k zastavení komunikace, nebo nedojde k chybě [1].

1.4.2 Autostart

Jde o funkci, která umožňuje spuštění spoje druhým koncem. Jeden konec se nakonfiguruje do režimu AutoStart, druhý se poté někdy zapne a začne vysílat NULL znaky. Příjem těchto znaků na první straně pak způsobí navázání komunikace a spuštění spoje. Do tohoto módu mohou být nastaveny oba konce, iniciovat spojení tedy mohou oba dva.

Díky této funkci není třeba zapínat oba konce zvlášť svými příslušnými zařízeními, což by vedlo ke ztrátám energie, kdy by jeden konec vysílal NULL znaky a druhý byl stále vypnutý [1].

1.4.3 Regulace toku dat

Protože přijímací moduly mají omezenou paměť (většinou FIFO paměť), je třeba regulovat tok přijatých dat, aby nedošlo k přetečení těchto pamětí a tedy ke ztrátě dat.

Vysílač proto může posílat N-znaky pouze tehdy, pokud přijímač má místo. Indikace toho probíhá posíláním FCT znaků. Za každý FCT znak, který jedna strana pošle, může od druhé obdržet 8 N-znaků. Těchto znaků může být posláno více po sobě, čemuž odpovídá vždy 8násobek možných N-znaků, které lze poslat. Toto číslo, počet kreditů, si musí vždy každý vysílač počítat sám. Pokud je tato hodnota vyčerpaná, může posílat pouze L-znaky. Při stavu *ErrorReset* je kreditový počet vždy nulový.

Rozhraní SpaceWire povoluje maximální počet 7 vyslaných FCT, tedy 56 N-znaků [2].

Výměnná vrstva také určuje prioritu jednotlivých druhů znaků:

1. Časový kód
2. FCT znak
3. N-znak
4. NULL znak

Tak je zaručeno, že časové kódy jsou poslány první a jejich distribuce po celé síti je co nejrychlejší. Poté následují FCT, které je třeba posílat podle místa v příchozích pamětích a je tedy nutné přerušovat tok N-znaků. Na posledním místě jsou NULL znaky, které se při normálním provozu používají pouze k udržení chodu spoje při absenci ostatních znaků [1].

1.4.4 Detekce a řešení chyb

V rámci výměnné vrstvy je definováno 5 chyb, jejichž výskyt zneplatní znakovou synchronizaci a regulaci datového toku. Proto je nutné rozhraní resetovat a inicializaci

provést znovu. To přinutí udělat to samé na druhém konci spoje (ukončení komunikace způsobí rozpojovací chybu – viz dole). Tento proces se nazývá výměna ticha, kdy jsou oba konce ve stavech *ErrorReset*, *ErrorWait* nebo *Ready* (viz **Obr. 4**), při kterých nejsou posílána žádná data. Poté může dojít k podání ruky a obnovení normálního provozu [1].

Rozpojovací chyba

Správně fungující SpaceWire spoj neustále přenáší data a proto se signály Data nebo Strobe nepřetržitě mění. Pokud od poslední změny těchto signálů uplyne přibližně 850 ns, přijímač to detekuje jako chybu.

Aby k této detekci došlo, je třeba nejprve přijmout aspoň jeden bit [2].

Chyba parity

Jakmile je paritní bit přijat, dojde k jeho kontrole okamžitě, protože paritní bit pokrývá bity, které přijdou před ním. V síti SpaceWire je použita lichá parita, jeho hodnota tedy musí být taková, aby v pokryté oblasti byl počet logických 1 lichý.

Pokud jeho hodnota není správná a již byl přijat první NULL znak, dojde k chybě parity [2].

Escape chyba

Escape (ESC) znak se používá pouze k vytvoření NULL nebo časového kódu, tedy je následován FCT znakem nebo jedním datovým znakem. Pokud je následován znakem jiným, dojde k escape chybě [2].

Kreditová chyba

Tato může nastat při běžném chodu, pokud je přijat N-znak, který není očekáván, což se odvíjí podle počtu odeslaných FCT znaků [2].

Chyba znakové sekvence

Tato chyba může nastat pouze při inicializaci spoje, pokud FCT znak přijde před odesláním NULL znaku, či je přijat N-znak, které by měly správně chodit pouze při normálním provozu [2].

Prázdné pakety

Jsou to pakety složené pouze ze znaku EOP nebo EEP, které následují po EOP nebo EEP minulého paketu. Protože tyto pakety nevznikají při normální operaci, jsou zahazovány [2].

Hlášení chyb síťové vrstvě

Pokud se rozpojovací, paritní, escape nebo kreditová chyba stane při normální operaci, tzn. ve stavu *Run*, informace o tom bude předána síťové vrstvě, aby se mohla provést další opatření a nedošlo ke ztrátě informací na spoji [2].

1.4.5 Distribuce systémového času

Probíhá pomocí časových znaků (viz 1.3), které se rozesílají po síti. Data v těchto znacích jsou tvořena 6bitovými čísly, proto všechny koncové body a směrovače mají 6bitové čítače. V celé síti pak musí být jedno rozhraní určené jako hlavní, které to řídí.

Periodicky ve zvolených intervalech, např. každou milisekundu, dostane hlavní rozhraní příkaz od svého zařízení. V tu chvíli inkrementuje svůj čítač a vyšle časový znak s touto hodnotou a dvěma bity kontrolních vlajek, nastavených podle vstupu ze zařízení.

Jakmile jiné rozhraní přijme tento znak, porovná hodnotu ve znaku a ve svém čítači, a pokud příchozí hodnota je o jednu větší (modulo 64), inkrementuje svůj čítač, signalizuje svému zařízení o příchodu nového časového znaku a předá mu hodnotu ovládacích vlajek. V případě směrovačů dojde také k přeposlání znaku do ostatních připojených rozhraní.

Pokud je příchozí hodnota stejná jako ta v čítači, nedojde k signalizaci místnímu zařízení, ani nedojde k rozeslání do dalších bodů sítě. To zabraňuje opakovanému posílání časových kódů po kruhové síti.

Při chybě na spoji, kdy rozhraní je ve stavu *ErrorReset*, dojde k vynulování čítače a výstupních hodnot ovládacích vlajek do zařízení.

V případě, že je příchozí hodnota menší, časový kód nebo hodnota čítače je neplatná, čítač je upraven na hodnotu příchozího kódu, ale není provedena signalizace místnímu zařízení, ani nedojde k rozšíření do dalších bodů sítě. To zabrání rozšiřování možného špatného časového kódu do zbytku sítě. Pokud je správný, při dalším kódu pak dojde k obnovení správné funkce. Znamená to však, že při chybě v síti se oprava šíří poměrně pomalu – při každém dalším kódu nastane oprava pouze sousedního bodu.

Z tohoto chování lze usoudit, že nelze v jednotlivých zařízeních v síti čas aktualizovat pouze časovými kódy, protože by v důsledku chyb v síti mohlo docházet k velkým nepřesnostem. Lepší je proto např. počítat čas místně a vždy při signalizaci příchozího kódu zkontrolovat, jestli místní čas je násobkem příchozí hodnoty.

Při posuzování přesnosti distribuce času je také nutné vzít v potaz zpoždění dané dobou vysílání časového znaku, která je 14 bitových period (ESC a datový znak je 4 + 10 bitů) a její absolutní hodnota závisí na přenosové rychlosti. Další zpoždění také

vzniká tím, že je v rozhraní třeba počkat, než skončí již probíhající vysílání znaku. Celkové zpoždění určitého bodu v síti pak záleží na jeho topologické vzdálenosti od původce systémového času [2].

1.5 Paketová vrstva

Tato vrstva definuje pakety, tedy bloky dat, kterými jsou informace posílány po síti. Skládají se ze tří částí, cílové adresy, nákladu a koncové značky.

Cílová adresa obsahuje 0 nebo více adresních položek, přičemž každá je tvořena jedním datovým znakem. Tato část paketu je důležitá hlavně pro členité sítě, kde je třeba data směřovat na správné místo. Pro spojení dvou bodů směrování není třeba, a proto je cílová adresa prázdná [1].

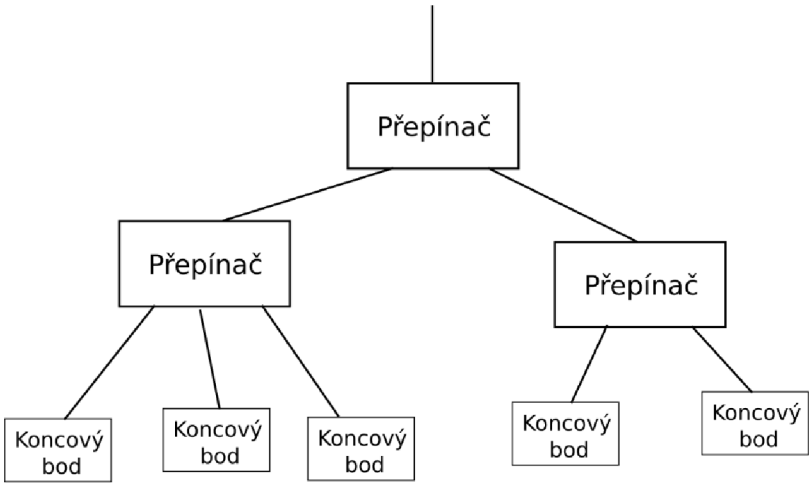
Náklad obsahuje jeden a více datových znaků. Jejich množství je neomezené a záleží na zařízením používající rozhraní.

Koncová značka paketu je tvořena znakem EOP nebo EEP. EOP se používá v normálních případech, kdy paket přišel v pořádku. EEP naopak značí, že při posílání paketu někde v síti došlo k chybě a data jsou neúplná [2].

1.6 Síťová vrstva

Zde jsou definována specifika sítí SpaceWire. Obvykle je třeba spojit více zařízení, jako jsou procesor, paměti či různé senzory. Tyto koncové body pak mezi sebou vedou komunikaci, o jejíž režii se stará tato vrstva.

Rozhraní koncových bodů mohou mít více připojitelných spojů, nicméně pro potřeby velkých sítí to nemusí stačit. Proto se používají směrovací přepínače, které mají mnoho portů a mohou spojovat mnoho zařízení, přičemž dokáží podle směrovacích adres předávat příchozí pakety na příslušné spoje. Tyto přepínače lze v případě potřeby také řetězit za sebou a spojit tak více zařízení, než které by dokázal obsloužit jeden přepínač (viz **Obr. 5**) [2].



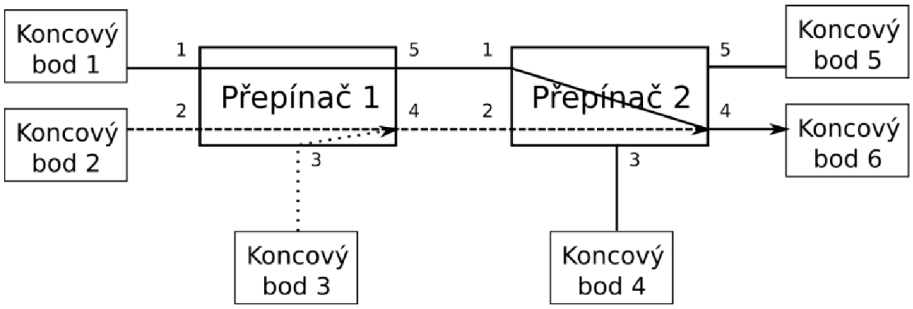
Obr. 5: Příklad sítě s přepínači

1.6.1 Směrování a řízení toku

Z pohledu síťové vrstvy jsou nejmenším kusem dat pakety, které jsou pro ni nedělitelné a jsou vždy přepravovány skrz síť jako celek. Řízení toku na jednotlivých spojích je pak řešeno výměnnou vrstvou (viz 1.4).

Protože velikost paketů není omezená, používá se tzv. wormhole směrování, kdy jakmile přepínač přečte adresu a rozhodne, který port má použít, začne okamžitě směrovat příchozí data do tohoto portu [2].

Jak již bylo řečeno, pakety jsou nedělitelné, proto ty dlouhé při průchodu sítí mohou zablokovat i několik spojů, což může zastavit pohyb jiných paketů (viz **Obr. 6**) [1].



Obr. 6: Blokování sítě dlouhým paketem

1.6.2 Adresace paketů

Probíhá pomocí adres uložených v první části paketu. Jejich podoba se liší podle metody použité k adresování, fyzické nebo logické adresování.

Při fyzickém adresování je na začátku paketu sekvence čísel, označující výstupní porty přepínačů, kterými bude paket procházet. Každý přepínač pak vždy z prvního čísla pozná port, na který má paket přesměrovat a poté toto číslo odmaže. Paket pak nemusí

obsahovat nepotřebná data a logika přepínače může být poměrně jednoduchá. Nevýhodou je, že v sítích s mnoha přepínači může adresa být velmi dlouhá.

Logická adresa naopak je krátká, je to unikátní číslo označující konkrétní koncové místo, do kterého se má paket dostat. Každý přepínač pak musí mít směrovací tabulku, podle které určí správný výstupní port. Ve velké síti může tato tabulka být celkem velká, proto se může síť rozdělit do několika oblastí, kdy jeden adresní znak je vždy určen pro jednu oblast. Adresa je tedy tak dlouhá podle toho, kolika oblastmi musí paket projít. Jakmile se paket dostane do další oblasti, odmaže se první adresní znak. Každý přepínač pak musí znát pouze adresy bodů své oblasti a adresy ostatních oblastí [1].

1.6.3 Detekce a řešení chyb

Na síťové vrstvě se mohou vyskytnout 3 druhy chyb - chyba spoje ohlášená výměnnou vrstvou, přijetí EEP nebo neplatná cílová adresa [2].

Chyba spoje ohlášená síťovou vrstvou

Pokud se tato chyba vyskytne ve zdrojovém nebo cílovém konečném bodu, bude ohlášena aplikační vrstvě, tedy zařízení připojenému k rozhraní. Na něm záleží, jak se vypořádá s touto chybou.

Dále se přeruší právě probíhající přenos paketu a N-znaky, které nebyly odeslány, se zahodí včetně koncového EOP nebo EEP znaku. Na přijímající straně spoje se pak za přijatá data přidá EEP znak [2].

Přijetí EEP

EEP na konci paketu značí, že při přenosu se vyskytla chyba, a data, které paket obsahuje, mohou být platná, ale jsou neúplná.

Takový paket se pak šíří skrz přepínače stejně jako paket zakončený EOP až do cílového bodu, kde je přítomnost tohoto znaku ohlášena aplikační vrstvě [2].

Neplatná cílová adresa

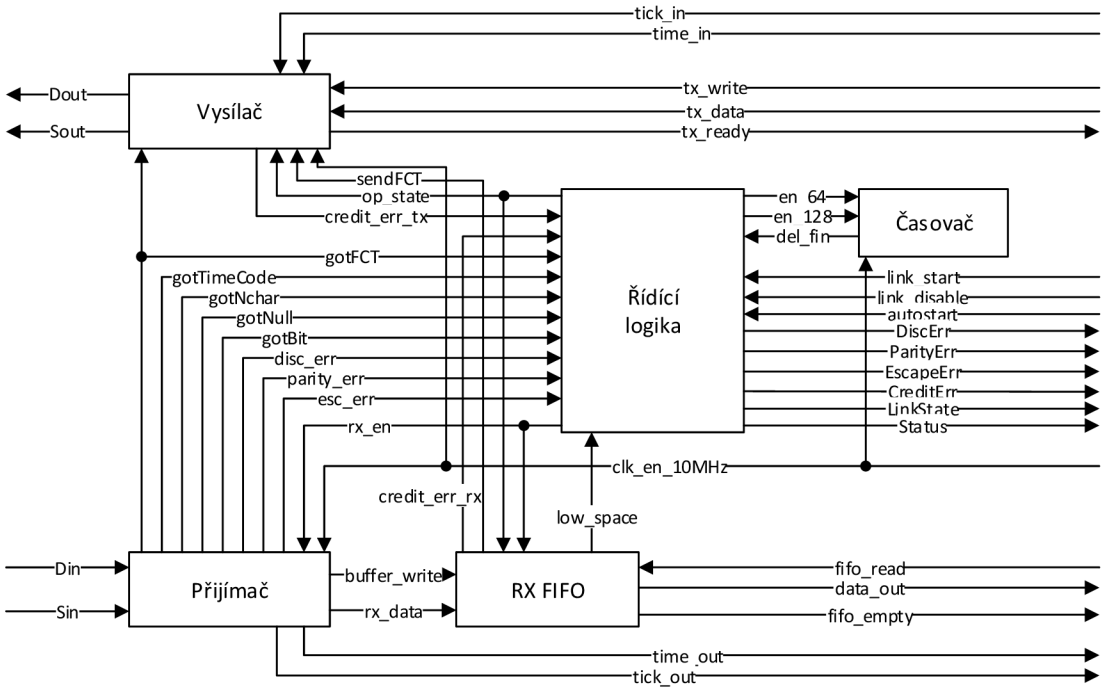
Pokud paket s neplatnou adresou přijde do přepínače nebo koncového bodu, dojde k jeho zahození [2].

2 Blokový návrh SpaceWire IP core

V této kapitole bude popsána implementace koncového bodu – jeho hierarchická struktura a detaily implementace jednotlivých funkcí.

Design vychází ze schématu prezentovaného ve standardu [2], které bylo upraveno pro konkrétní podobu rozhraní. Další rozdíly vyplývají z toho, že standard u některých funkčních bloků nespecifikuje přesné provedení, ale pouze výsledné chování.

Celý obvod kromě části přijímače je časován hodinovým signálem nadřazeného systému. K časování některých dějů s pevnou časovou specifikací rozhraní dále vyžaduje povolovací pulz s frekvencí 10 MHz, který je odvozený od systémového hodinového signálu a jehož šířka je jedna perioda tohoto signálu.



Obr. 7: Blokové schéma rozhraní

V blokových diagramech nejsou zobrazené signály *clk* (systémový hodinový signál) a *rst* (systémový synchronní reset), protože jsou vedené do všech součástí obvodu. Výjimkou je komponenta *RX Front*, nacházející se v *Přijímači*, která nevyužívá systémový hodinový signál.

Časová rozhraní jsou implementována tak, že dokáží vysílat na příkaz obdržený časový kód od aplikace a naopak přijmout příchozí časový kód a předat jej aplikaci.

Samotné čítače určené k synchronizaci času si však nadřazený systém musí implementovat sám.

Styl kódu

Kód je psaný tak, že jsou oddělené sekvenční a kombinační části. Použité jsou procesy. Mimo ně se pak vyskytuje některá kombinační logika, přiřazení výstupů a mapování komponent.

Jména klopných obvodů mají koncovku *_reg*, jejich vstupy pak *_nxt*. Signály použité ke spojení dvou komponent jsou zakončené *_s*.

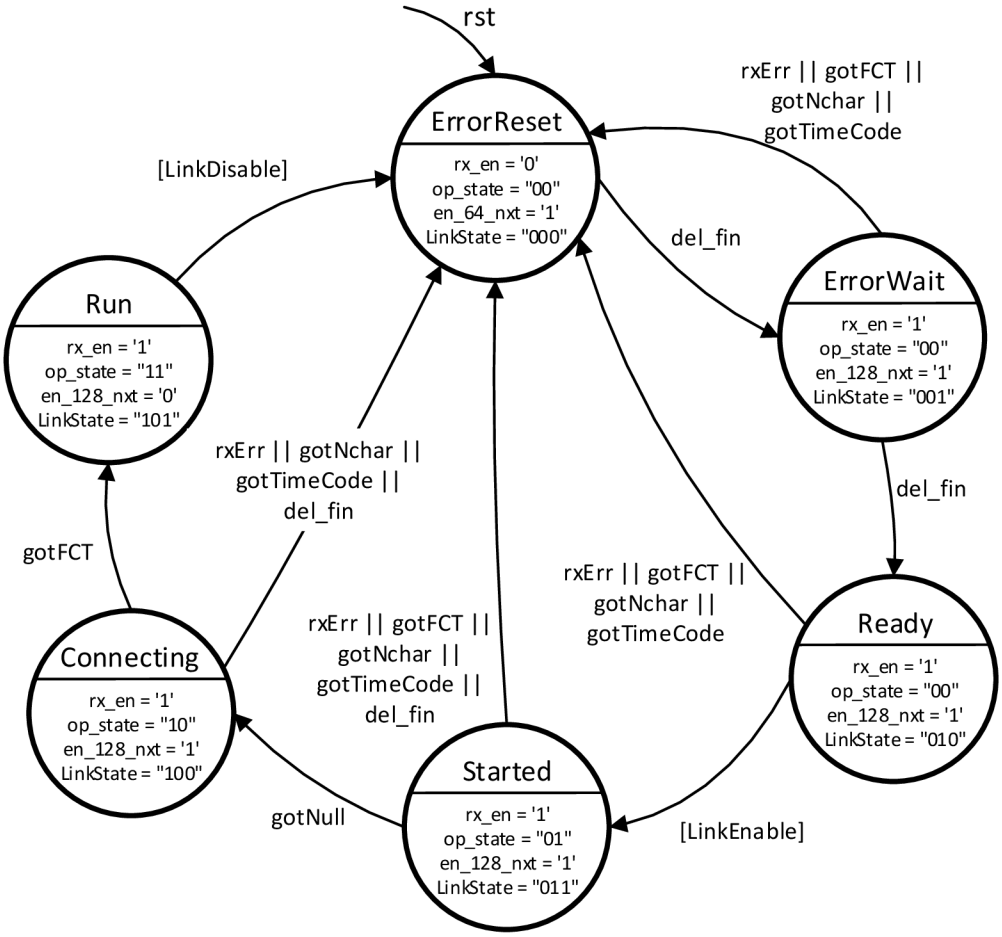
Návrh je rozdělen do 12 souborů s VHDL kódem. 11 tvoří bloky a podbloky popsané v následujících kapitolách. Poslední, *top.vhd*, slouží k jejich spojení tak, jak je to zobrazeno na **Obr. 7**. Přiloženo je také 5 simulačních VHDL souborů. 4 slouží k dílčí simulaci základní funkcionality. Pátý, pojmenovaný *top_tb.vhd*, je použit pro simulaci popsanou v 3.1.3.

2.1 Řídící logika

Řídící logika reprezentuje výměnnou vrstvu standardu (viz 1.4). Řídí průběh operace celého rozhraní při inicializaci linky. Dále komunikuje s aplikační vrstvou, kterou je ovládána, a které poskytuje informace o aktuálním stavu rozhraní a nastalých chybách.

Hlavní činnost komponenty, tedy inicializace linky, tzv. handshake, je provedena stavovým automatem (viz), jenž vychází ze stavového diagramu popsaného ve standardu [2]. Pro časová zpoždění, definovaná pro inicializaci a timeout linky, je využit *Časovač* (viz 2.2). Ten je aktivován pulzem *en_64* pro zpoždění 6,4 µs, nebo *en_128* pro zpoždění 12,8 µs. Pulz je generovaný pomocí klopného obvodu použitého jako detektor náběžné hrany. Po uplynutí požadovaného zpoždění pak přijde pulz na vstupu *del_fin*, který iniciuje přechod *Řídící logiky* do jiného stavu a zároveň je resetován klopný obvod pro spouštění *Časovače*.

Dále jsou pro Řídící logiku důležité informace o přijatých znacích a nastalých chybách v *Přijímači*, *RX Bufferu* a *Vysílači*. Přijaté znaky a kódy mohou být NULL (*gotNull*), FCT (*gotFCT*), N-znak (*gotNchar*) a časový kód (*gotTimeCode*). Indikován je také datový bit (*gotBit*). Přijetí je indikováno jednou periodou dlouhým pulzem odpovídajícího signálu. Výjimkou jsou signály *gotBit* a *gotNull*, které jsou po přijetí prvního bitu, respektive NULL kódu, v logické 1. Indikace chyb vždy probíhá jednu periodu dlouhým pulzem. Tyto chyby na lince jsou detekovány v *Přijímači* – chyba rozpojení (*disc_err*), chyba parity (*parity_err*), chyba znakové sekvence (*esc_err*) a kreditové chyby z *Vysílače* (*credit_err_tx*) a *RX Bufferu* (*credit_err_rx*).



Obr. 8: Stavový automat

V rámci inicializace musí být *Řídící logika* schopna resetovat některé bloky. K tomu využívá povolovací signál *rx_en* k řízení *Přijímače* a některých částí *RX Bufferu*. Protože některé klopné obvody musí používat asynchronní reset (*RX Char* a vstupy *Kontroly rozpojení*), je tento výstup veden přes klopný obvod (*rx_en_reg*), aby nemohlo dojít k náhodnému resetu vlivem zákmitů. *Vysílač* nelze pouze zapínat či vypínat, je třeba mu také sdělovat, jaké znaky může v dané chvíli posílat. Tyto informace jsou zakódované v signálu *op_state* (viz). Tento signál je veden také do *RX Bufferu*, aby bylo poznat, zda chyba nastala v *Run* stavu.

Komunikace s aplikací

Řídící logika umožňuje nadřízené aplikaci rozhraní ovládat pomocí tří signálů. Jsou to signály *LinkStart*, *LinkDisable* a *Autostart*, které slouží k zapnutí, vypnutí, respektive k aktivaci *AutoStartu* (viz 1.4.2).

Pro přechod ze stavu *Ready* do stavu *Started* musí být splněna podmínka [LinkEnable], která je definována dle rovnice (1).

$$LinkEnable = \overline{LinkDisable} * \overline{low\_space} * (LinkStart + (AutoStart * gotBit)) \quad (1)$$

Signál *low_space* pochází z *RX Bufferu* a je v logické 1, pokud je v paměti pro příchozí data místo méně než pro 8 znaků. V takovém případě je pozastaveno spuštění linky, protože by si rozhraní nemohlo vyžádat zaslání N-znaků a běželo by zbytečně. U složitějších sítí by navíc mohlo dojít k zablokování části sítě (viz 1.6.1).

Po začátku handshaku pak aplikace může rozhraní vypnout signálem *LinkDisable* až ve stavu *Run*.

Řídící logika předává informace o nastalých chybách nadřazenému systému, pokud k nim dojde při normální operaci, tedy ve stavu *Run* (viz 1.4.4). Jde o signály *DiscErr* (chyba rozpojení), *ParityErr* (chyba parity), *EscapeErr* (chyba znakové sekvence) a *CreditErr* (kreditová chyba). Dále také informuje o stavu, ve kterém se blok právě nachází. Stavy jsou binárně kódované do 3 bitového signálu *LinkState*. Aplikace také může využít 10bitového signálu *Status* (viz **Tab. 1**), který sdružuje další informační signály. Protože se jedná o vývody interních signálů navrženého rozhraní, chyby jsou tak indikovány ve všech stavech.

Tab. 1: Napojení signálů na výstup Status

Status	příslušný signál
Status(0)	disc_err
Status(1)	parity_err
Status(2)	esc_err
Status(3)	credit_err_rx
Status(4)	credit_err_tx
Status(5)	gotBit
Status(6)	gotNull
Status(7)	gotFCT
Status(8)	gotNchar
Status(9)	gotTimeCode

2.2 Časovač

Komponenta sloužící k odměřování zpoždění pro *Řídící logiku*. Generovaná zpoždění mají dvě možné doby trvání – 6,4 μs (5,82 μs až 7,22 μs, viz. níže) a 12,8 μs (11,64 μs až 14,33 μs). V jednu chvíli je potřeba počítat vždy jen jedno zpoždění, a proto

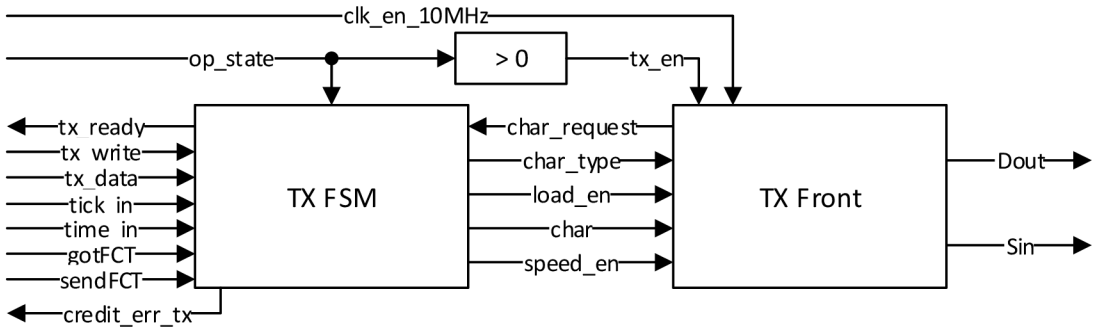
je použit jeden čítač. Jedná se o 7bitový dekrementující čítač, který snižuje svou hodnotu o 1 s aktivní hodnotou povolovacího signálu odvozeného ze systémového hodinového signálu o kmitočtu 10 MHz (*clk_en_10MHz*). Díky tomu je počet dekrementací vždy 63 nebo 127 a nezáleží na systémovém hodinovém signálu. Dosažení 0 je pak signalizováno signálem *del_fin*.

Standardem povolené časové rozsahy jsou odvozené z 10 MHz signálu, 5,82 µs až 7,22 µs je 64 period 11 MHz až 65 period 9 MHz hodinového signálu a 11,64 µs až 14,33 µs je 128 period 11 MHz až 129 period 9 MHz hodinového signálu [2]. Proto je výhodné použít pro počítání právě 10 MHz povolovací signál.

2.3 Vysílač

Obstarává funkce znakové (viz 1.3) a částečně signálové vrstvy (viz 1.2.3). Hlavní funkcí je odesílání znaků přes linku. Ty jsou vybírány podle stavu *Řídící logiky* (indikované signálem *op_state*) a poté podle priority znaků dané standardem (viz 1.4.3). Vysílání může probíhat ve dvou rychlostech – nižší na frekvenci 10 MHz, která je poskytována skrz povolovací signál *clk_en_10MHz* odvozený od systémového hodinového signálu, používaná při inicializaci, a vyšší rychlost na frekvenci systémového hodinového signálu užitá pro normální provoz linky (stav *Run*).

Vysílač sestává ze dvou komponent – *TX FSM* a *TX Front*, viz **Obr. 9**. První slouží k výběru a přípravě dalšího znaku k poslání. Druhý se stará o samotnou serializaci znaku a jeho odeslání po lince.



Obr. 9: Blokové schéma vysílače

2.3.1 TX FSM

Jak už bylo zmíněno, blok *TX FSM* slouží primárně k výběru nového znaku k odeslání. Výběr je proveden vždy, když si blok *TX Front* žádá další znak, tedy signál *char_request* je v logické 1. Jádrem bloku *TX FSM* je stavový automat, který má 3 stavy – *general*, *null_fct*, *time_char*. Ovládací kódy, tedy sekvence 2 znaků (viz 1.3.1), je třeba poslat za sebou. Proto je zde stav *null_fct*, který slouží k poslání FCT za ESC znakem

k vytvoření NULL kódu, a *time_char* stav k poslání N-znaku za ESC znakem k vytvoření časového kódu. Většinu času se stavový automat vyskytuje ve stavu *general*, kdy výběr znaku probíhá podle hodnoty signálu *op_state* a následně podle priority druhů znaků. Při vyžádání dalšího znaku je tedy na výstup *char_out* nastaven zvolený znak, na výstup *char_type* jeho typ a zároveň je toto indikováno logickou 1 na signálu *load_en*.

Pro určení paritního bitu blok využívá klopný obvod *parity_reg*, jehož hodnota značí lichou paritu části předchozího znaku, která se týká paritní oblasti zasahující do nynějšího znaku (viz 1.3). Podle této hodnoty tak lze určit paritní bit, přičemž se zároveň nastaví nová úroveň tohoto klopného obvodu.

Ve stavu *general* a při hodnotě *op_state* „11“ přechází výstup *speed_en* do logické 1, což způsobí přepnutí rychlosti na nominální rychlost linky a odpovídá frekvenci systémového hodinového signálu. Přepínání rychlosti vysílání v *TX FSM* zajišťuje, že ke změně rychlosti dochází pouze po odeslání celého znaku či kódu.

Načítání a příprava časových kódů a N-znaků probíhá paralelně s funkcí stavového automatu. Když chce aplikace poslat časový kód, nastaví jeho hodnotu na vstup *time_in* a zároveň nastaví signál *tick_in* na logickou 1 po dobu jednoho hodinového taktu. Obvod tento časový znak uloží do interního registru a pomocí klopného obvodu *tick_in_reg* nastaveného do logické 1 sdělí stavovému automatu, že jej má poslat. Stavový automat po odeslání znaku nastaví tento klopný obvod opět do logické 0. Pokud chce aplikace poslat další časový znak, aniž by starý byl poslán, dojde k přepsání starého znaku. Protože časové kódy lze posílat pouze ve stavu *Run*, nemá smysl si chystat časový kód, který zatím nelze odeslat, a proto mimo stav *Run* je signál *tick_in* ignorován.

Příprava N-znaků probíhá podobně. *Vysílač* neobsahuje FIFO na uložení více znaků a může si ze vstupu *data_in* do registru *nchar_reg* přichystat vždy jen jeden N-znak. Aby mohl načíst další, musí být předchozí znak odeslaný (klopný obvod *nchar_ready_reg* je v logické 0) a počet kreditů (uložený v 6bitovém registru *credit_cnt_reg*) musí být větší než 0. Při odeslání N-znaku je pak z tohoto čítače odečtena 1. Přičítání se naopak děje, když *Přijímač* přijme FCT znak (*gotFCT)* a *op_state* se rovná „11“ (hodnota *op_state* je vyžadována z důvodu, že první FCT je využito pouze na inicializaci linky). Pokud hodnota čítače není větší než 48, přičte se 8, jinak dojde ke kreditové chybě (*credit_Err*).

TX FSM zajišťuje posílání kódu FCT, čímž indikuje druhé straně volné místo v přijímacím FIFO a možnost přijímat N-znaky. FCT kódů může být poslán vždy pouze určitý počet, který určuje *RX Buffer*. Ten pomocí signálu *sendFCT* indikuje počet FCT tak, že za každou periodu, kdy je tento signál v logické 1, má být poslán 1 FCT znak. *TX FSM* si tento počet ukládá do 4 bitového čítače. Při odeslání FCT se pak provede

dekrementace. Počáteční hodnota tohoto čítače je 1, z důvodu již zmíněného prvního FCT použitého pouze na inicializaci.

2.3.2 TX Front

Druhá část *Vysílače* se stará o odesílání znaků po jednotlivých bitech spolu s hodinovým signálem zakódovaným do signálu Strobe. K tomu využívá posuvný registr *shift_reg*, ze kterého jsou bity vysouvány do výstupního klopného obvodu *dout_reg*. Porovnáním jeho hodnoty a hodnoty posledního bitu posuvného registru lze zjistit, zda se má úroveň signálu Strobe, vystupující z klopného obvodu *sout_reg*, změnit.

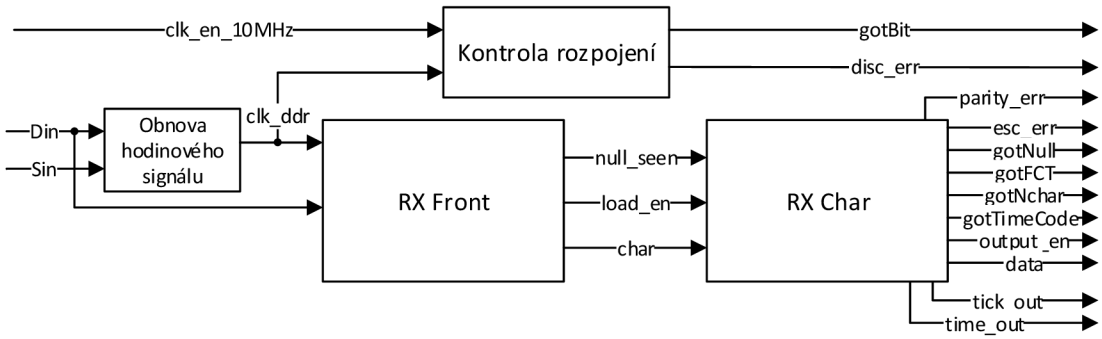
Blok si nejprve vyžádá znak signálem *char_request*. Pokud je na vstupu *load_en* logická 1, dojde k načtení znaku ze vstupu *char_in* do posuvného registru. Přitom dojde podle hodnoty signálu *char_type*, značícího typ znaku, k nastavení dekrementačního čítače *cnt_reg* na 4 nebo 10. Ten je použit k odměření délky znaku. Jakmile je na hodnotě 1, zbývá již vysunout na výstup poslední bit. V tu chvíli si komponenta vyžádá další znak. Pokud jej obdrží, nastaví čítač opět na novou hodnotu a posílá dál. Pokud ne, čítač se zastaví na hodnotě 0 a hodnota *sout_reg* se přestane měnit. Toho se využívá také před posláním prvního znaku, kdy čítač má po resetu hodnotu 0, díky čemuž nejsou odesílány nulové bity před načtením prvního znaku do posuvného registru.

Vysílání může probíhat při dvou rychlostech (viz nahoře). Impulz k přepnutí z nižší na vyšší rychlost dává *TX FSM* skrz signál *speed_en*. *TX Front* na to může reagovat pouze při načtení nového znaku. Protože je předtím třeba odeslat ještě poslední bit předchozího znaku, je přepnutí zpožděno o 2 takty, místo o 1. První se tedy logická 1 zapíše do *speed_en_aux_reg* a až potom do *speed_en_reg*, ze kterého se rychlost vyvozuje. Protože se pomalejší rychlost používá jen při inicializaci, jinak než resetem bloku z vyšší na nižší rychlost přepnout zpět nelze.

Protože při resetu nesmí být přechod logických úrovní zároveň na *dout_reg* a *sout_reg* (viz 1.3.3), musí zde být klopný obvod *en_delay_reg*, pomocí kterého je reset *dout_reg* o jeden takt zpožděn.

2.4 Přijímač

Přijímač reprezentuje znakovou vrstvu (viz 1.3). Hlavním účelem je příjem a zpracování dat. O jejich typu informuje *Řídící logiku*. N-znaky jsou ukládány do *RX Bufferu*. Časové kódy jsou předávány aplikaci.



Obr. 10: Blokové schéma přijímače

Blok se skládá ze 4 komponent. *Obnova hodinového signálu* rekonstruuje hodinový signál ze vstupních signálů *Din* (Data) a *Sin* (Strobe), který je pak použit ke zpracování samotných dat v bloku *RX Front*. Ten převede vstupní sériová data na paralelní v podobě jednotlivých znaků. Ty jsou následně přeneseny do komponenty *RX Char*, kde jsou synchronizovány na systémový hodinový signál a zpracovány. Poslední je *Kontrola rozpojení*, jež kontroluje aktivitu na vstupech, a v případě neaktivity ohlásí rozpojovací chybu.

2.4.1 Obnova hodinového signálu

Rekonstruuje hodinový signál ze vstupů *Din* a *Sin* pomocí hradla XOR (nonekvivalence). Výstupem je signál *clk_ddr*.

2.4.2 RX Front

Komponenta paralelizující vstupní data. Jako jediná v designu není časovaná systémovým hodinovým signálem, nýbrž obnoveným hodinovým signálem *clk_ddr*. Protože není zaručena přítomnost tohoto signálu v každé chvíli, je v tomto bloku použit asynchronní reset (*rst*),

Signál *clk_ddr* má poloviční frekvenci oproti původnímu signálu použitému k odeslání dat z druhé strany. Proto je třeba vstupní datový signál (*Din*) vzorkovat na náběžnou i sestupnou hranu signálu klopnými obvody *data_rise* a *data_fall*. Kromě *data_fall* jsou ostatní klopné obvody citlivé na náběžnou hranu.

Za nimi je zapojený 11bitový registr shift_reg, který je zapojen jako dva posuvné registry, 5bitový za *data_rise* a 6bitový za *data_fall*. Tímto způsobem každou náběžnou hranou hodinového signálu dojde k načtení páru bitů, který je postupně posunován registrem. Nejprve jsou data načítána volně, přičemž je hledána sekvence „011101000“, což je NULL kód a paritní bit. Její detekce umožní synchronizaci přijímače vůči vstupním datům, kdy pak přesně ví, kolikátý bit znaku právě načítá. Protože jsou však data načítána po dvojicích, může být hledaná sekvence o bit posunutá, a proto je třeba testovat dva

rozsahy ze *shift_reg*. Pokud znak, jenž má vždy sudý počet bitů, obsazuje v bitovém páru vždy oba bity, zapíše se logická 1 do klopného obvodu *null_even_reg*. Pokud je naopak znak o bit posunutý a v bitovém páru, kde jeden bit je konec jednoho znaku a druhý bit začátek druhého znaku, je zapsána logická 1 do klopného obvodu *null_odd_reg*. Od těchto řídících klopných obvodů je odvozena činnost zbytku bloku. Závisí na tom jak poloha typového bitu (druhý bit znaku, udává typ znaku) v registru, tak také zapojení výstupů ze *shift_reg* do registru *char_reg*.

K počítání bitových párů je použit 3bitový čítač *cnt_reg*. Podle typu znaku počítá od 0 buď do 1 (L-znak) nebo do 4 (N-znak). Poté dojde k předání načteného znaku do *char_reg*, kdy v případě L-znaku jsou zbylé bity doloženy logickými 0. Zároveň je také na jednu periodu nastaven klopný obvod *en_reg*, který slouží jako povolovací signál pro přesun mezi hodinovými doménami.

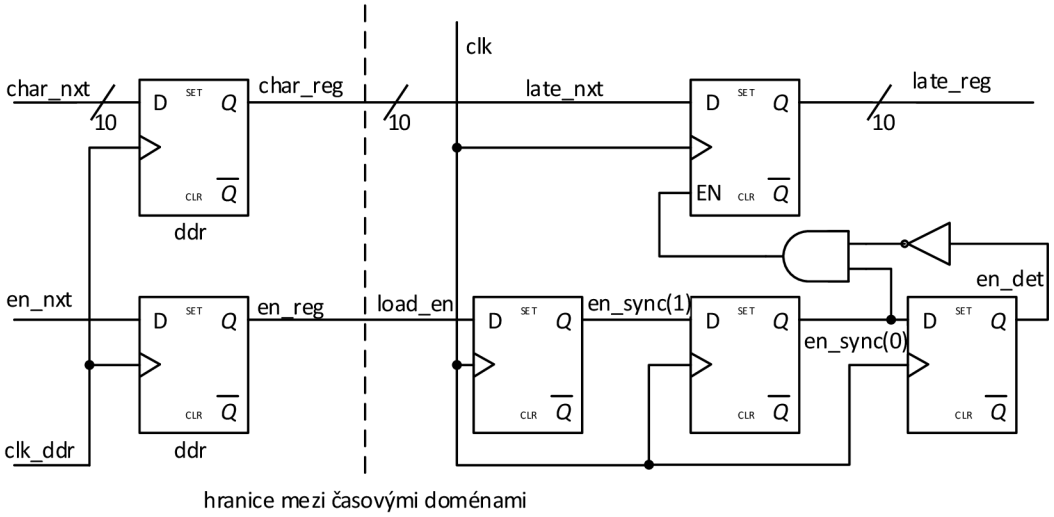
2.4.3 RX Char

Komponenta vyhodnocující přijaté znaky. Testuje jejich platnost. Jejich typ indikuje *Řídící logice*. N-znaky pak předává *RX Bufferu* a časové kódy aplikaci.

Při přechodu mezi časovými doménami mohou vznikat na klopných obvodech metastabilní stavy porušením doby předstihu nebo přesahu, které by se šířily dál do obvodu. Vstupní data z *RX Front* je proto třeba synchronizovat zařazením několika klopných obvodů za sebou, aby metastabilní stavy měly čas se ustálit. Nemusí ale skončit na správné úrovni, proto u vícebitových signálů není jisté, zda jsou vstupní data správná. Řešením je synchronizovat pouze povolovací signál, u kterého by v případě ustálení v logické 0 byla načtena správná hodnota v dalším taktu. Protože synchronizace způsobuje zpoždění, je zajištěno, že datové signály již budou stabilní [13]. Použité jsou dva klopné obvody jako synchronizér (*en_sync*) a třetí pro detekci náběžné hrany (*en_det*).

Synchronizovat je také třeba vstup *null_seen* (pomocí *null_sync*), který je jako *gotNull* předán *Řídící logice* a informuje o přijetí prvního NULL kódu.

V rámci hlavní funkce komponenty je třeba vyhodnotit vlastnosti načtených dat. Protože pokrytí parity sahá přes dva znaky (viz 1.3), a po ESC mohou následovat pouze některé druhy (viz 1.3.1), je třeba mít v modulu vždy dva znaky naráz. V případě výskytu chyby pak dojde k jejich zahození.



Obr. 11: Schéma znázorňující přechod mezi časovými doménami

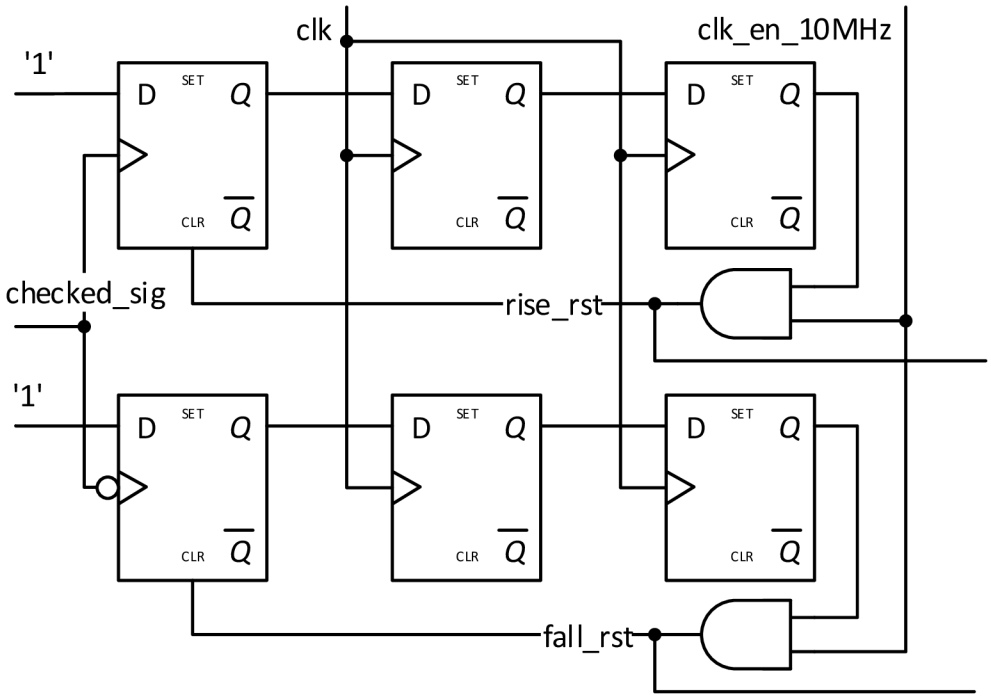
Při detekci náběžné hrany povolovacího signálu (viz) je do registru *late_reg* načten nový znak a zároveň předchozí je posunut do registru *early_reg*. Na jednu periodu hodinového signálu je nastaven do logické 1 klopný obvod *check_reg*, díky čemuž je zajištěno, že zpracování znaku proběhne pouze jednou. Nejprve je zkontrolována parita. Použita jsou hradla XOR, které jsou seskládány pomocí závorek v kódu tak, aby celkové zpoždění bylo co nejmenší. Poté je vyhodnocen typ znaku, a zda není porušena ESC sekvence. Podle typu znaku dojde k pulzu buď signálu *gotFCT* (FCT), *gotNchar* (N-znak, EOP, nebo EEP) nebo *gotTimeCode* (N-znak po ESC znaku). V druhém případě je také do klopného obvodu *char_typ_nxt* uložena logická 1. V případě časového kódu je logická 1 uložena do *time_type_nxt*.

Nakonec, pokud nedošlo k chybě, je podle minulých hodnot těchto klopných obvodů znak uložený v *early_reg* předán do FIFO (*char_type_reg* = ‚1‘) skrz výstup *data_out* (indikováno signálem *output_en*) nebo je předán aplikaci jako časový kód (*time_type_reg* = ‚1‘) výstupem *time_out*, jako signalizace je použit signál *tick_out*.

Znak je v registru *late_reg* zpracováván jako 10bitový. Do *early_reg* je předán jako 9bitový bez paritního bitu, který již není potřeba. Jako časový kód pro aplikaci je posláno horních 8 bitů. Znaky předávané FIFO paměti musí obsahovat typový bit (na pozici nejméně významného bitu), proto je použito všech 9 bitů. Datové znaky (typový bit je nulový) jsou uloženy nezměněny, EOP a EEP znaky jsou dle standardu uloženy tak, že typový bit je v logické 1, následující bit je buď v logické 0 (EOP) nebo v logické 1 (EEP) a všechny ostatní bity jsou v logické 0 [2].

2.4.4 Kontrola rozpojení

Kontroluje aktivitu linky a v případě nečinnosti zahlásí chybu rozpojení. Nečinnost je zde definována jako nezměněné úrovně vstupů Data a Strobe po dobu 850 ns (727 ns až 1000 ns - 8 period 11 MHz až 9 period 9 MHz hodinového signálu) [2]. Protože obnovený hodinový signál *clk_ddr* reflektuje změny na obou vstupech, stačí sledovat pouze ten. Zde však hrozí, že by signál byl kontrolovaný vždy, když má stejnou hodnotu. Došlo by tak k chybě i v případě aktivní linky. Proto jsou změny detekovány pomocí dvou klopných obvodů, kdy jeden je citlivý na náběžnou hranu a druhý na sestupnou. Nejsou ale časované systémovým hodinovým signálem, ale přímo sledovaným *clk_ddr* (v tomto bloku jako *checked_sig*). Díky tomu je změna zaznamenána vždy. Výstupy těchto klopných obvodů je pak třeba synchronizovat stejným způsobem jako v *RX Char*. V případě zaznamenané aktivity jsou pak odpovídající detekční klopné obvody asynchronně resetované (viz **Obr. 12**).



Obr. 12: Schéma detekce změn vstupního signálu

Čas je odměřený pomocí 4 bitového čítače, který je spouštěn 10 MHz povolovacím signálem (perioda 100 ns). Při neaktivitě je přičtena 1, při aktivitě je čítač resetován. V případě napočítání 8 je zahlášena rozpojovací chyba. Celkový odměřený čas neaktivity se tedy pohybuje mezi 800-900 ns, záleží, kdy dojde k poslední změně.

2.5 RX Buffer

Blok určený k ukládání přijatých N-znaků do té doby, než si je převezme nadřazená aplikace. Obsahuje paměť typu FIFO implementovanou do blokové paměti v FPGA. Její velikost je možné zvolit skrz generikum ADDR_LENGTH, které určuje bitovou šířku paměťového ukazatele. Hloubka paměti, uložená v konstantě MEM_DEPTH, je pak 2^ADDR_LENGTH. Velikost jednotlivých znaků je 9 bitů.

Pro přístup do paměti jsou použity dva registry jako ukazatele, *write_addr* pro zápis a *read_addr* pro čtení. Paměť tedy musí být dvouportová, aby umožňovala simultánní zápis a čtení. Obojí probíhá synchronně, zápis při povolovacím signálu v logické 1 (*we*). Data na výstupu objevují každý takt, čtením se rozumí až jejich načtení nadřazenou aplikací. Zápis probíhá na adresu danou registrem *write_addr_reg*. Protože i čtení je synchronní, je třeba použít vstup do čtecího registru (*read_addr_nxt*) místo výstupu (*read_addr_reg*), jinak by se na výstup data dostávala o takt později a čtení by nemohlo probíhat nepřerušeně.

Při zápisu nebo čtení je k odpovídajícímu ukazateli přičtena 1. Protože je velikost paměti odvozena od velikosti ukazatele, po dosažení maximální hodnoty dojde k přetečení ukazatele. To, který z nich má větší hodnotu, je třeba zaznamenat klopným obvodem *looped_reg*, kdy při logické 0 je *write_addr_reg* větší než *read_addr_reg* a při logické 1 je to naopak. Lze tak určit, zda je při shodnosti hodnot ukazatelů FIFO prázdné (*looped_reg* = ‚0‘), nebo plné (*looped_reg* = ‚1‘).

Tok dat ve standardu je řízen znaky FCT, kterými je indikována druhé straně schopnost přijmutí 8 N-znaků. To záleží hlavně na místu v paměti, proto příkaz k poslání FCT je podáván zde. Spočítáno je vždy volné místo a uloženo do registru *free_space_reg*. Buď je odečten rozdíl zapisovacího a čtecího ukazatele od MEM_DEPTH (*looped_reg* = ‚1‘), nebo je odečten zapisovací od čtecího ukazatele. Tato operace nebere v potaz změnu polohy ukazatelů v době výpočtu, proto je k výsledku přičtena 1 v případě čtení a odečtena 1 v případě zápisu. Počet vyžádaných znaků je uložen do registru *req_chars_cnt_reg*. Pokud není tato hodnota větší než 48 a volné místo je větší nebo rovno počtu vyžádaných znaků + 8, je vydán příkaz k poslání FCT (signál *sendFCT* nastaví na 1 takt do logické 1) a k počtu vyžádaných znaků je přičteno 8.

Protože jsou příchozí data řízená skrz FCT, není třeba *Přijímači* sdělovat, zda je možný zápis. Ten pouze signalizuje skrz signál *buffer_write* přítomnost dat na vstupu *data_in*. Ta jsou následně uložena do paměti a od počtu vyžádaných znaků je odečtena 1. Pokud není volné místo, dojde ke kreditové chybě.

Čtení znaků a jejich předávání aplikaci již vyžaduje oboustrannou režii. Pokud jsou na výstupu *data_out* platná data, výstup *fifo_empty* je v logické 0. Pokud chce pak aplikace vyčítat data z paměti a *fifo_empty* je v logické 0, nastaví signál *fifo_read* do logické 1 a načte znak. Protože však po zápisu znaku do prázdné paměti trvá jeden takt, než jsou na výstupu platná data, je změna úrovně *fifo_empty* z logické 1 do 0 zpozděna pomocí klopného obvodu *fifo_empty_reg*. Při vyprázdnění paměti to ale neplatí, přechod do logické 1 musí být bez zpoždění. Dosaženo je to odvozením od vstupu místo výstupu registru obsahující volné místo (*free_space_cnt_nxt*).

Reset

Tento blok skladuje přijatá data, než si je převezme aplikace. Proto obsah paměti není ovlivněn resetem přijímače (*rx_en* = ‚0‘), jediný registr ovlivněný tímto signálem je počet vyžádaných znaků. Pokud k resetu linky dojde ze stavu *Run* (přechod detekován pomocí signálu *op_state* a jeho o takt zpožděné verze *op_state_del*), na konec FIFO je vložen znak EEP. K tomu je použit klopný obvod *eep_insert_reg*. K resetu se váže také signál *low_space*, který je v logické 1, pokud je volné místo v paměti menší než 8. Ten zabraňuje startu linky (viz 2.1). Bokem stojí celosystémový reset (*rst*), který se vztahuje na celý blok. FIFO však vymazáno není, pouze se vynulují ukazatele.

3 Verifikace a syntéza

3.1.1 Funkční verifikace

Při vývoji byla ověřena základní funkcionalita pomocí komplexního verifikačního prostředí, která ale není součástí této práce. Dílčí bloky byly simulovány s pomocí jednoduchých testbenchů pro ověření základních funkcí. Dále byla provedena post-route simulace pro ověření časových parametrů DDR registrů spolu s rekonstruovaným hodinovým signálem. Tato simulace je podrobněji rozebrána v kapitole 3.1.3.

3.1.2 Syntéza

Návrh byl syntetizován pro FPGA Spartan-3 XC3S200 od výrobce Xilinx nástojem XST, který je součástí návrhového prostředí ISE 14.7.

Z důvodu použití sítě SpaceWire ve vesmíru je pro větší spolehlivost použita bezpečná implementace stavových automatů (přepínač *-safe_implementation*). Cenou za to je logika navíc. Dále bylo nastaveno kódování automatů na One Hot (*-fsm_encoding*). Každý stav je tak kódován jako jeden bit. Nutný počet klopných obvodů je větší, je ale dosaženo vyšší rychlosti. Syntéza proběhla bez chyb a varování. Výsledné využití prostředků je v **Tab. 2**.

Tab. 2: Využití zdrojů v Spartan-3 XC3S200

	Použité	Dostupné	Využití čipu
Počet logických bloků	234	1920	12 %
Počet klopných obvodů	201	3840	5 %
Počet 4vstupových náhledových tabulek	435	3840	11 %
Počet blokových RAM pamětí	1	12	8 %

To platí v případě využití blokové paměti RAM v FPGA. V případě použití distribuované paměti by vzrostlo využití náhledových tabulek. Konkrétní množství závisí na zvolené velikosti paměti dané parametrem ADDR_LENGTH.

Maximální kmitočet vstupních dat je omezen na přibližně stejnou hodnotu, jako je systémový hodinový signál. L-znak má 4 bity. Při přechodu mezi časovými doménami je třeba vzorkovat povolovací signál dvakrát pro jistotu jeho načtení do synchronizéru. Poté musí být načten do druhého klopného obvodu tvořící synchronizér. Protože načtení dalšího znaku do RX Char je řízeno detekcí náběžné hrany, je třeba stejným způsobem spolehlivě načíst také logickou 0. To trvá 4 periody hodinového signálu.

Syntéza byla provedena také pro FPGA RTAX 1000S od výrobce Microsemi. Použit byl nástroj Synopsis Synplify Pro G-2012.09A-SP4 a vývojové prostředí Libero IDE. Tento obvod se vyznačuje radiační odolností a je certifikován pro lety do vesmíru. Využití prostředků je uvedeno v **Tab. 3**. Minimální dosažitelná perioda systémového hodinového signálu je 11,3 ns, což odpovídá kmitočtu 88,8 MHz. Tyto hodnoty jsou však pouze přibližné, protože zpoždění propojení je při syntéze odhadováno.

Tab. 3: Využití zdrojů v RTAX 1000S

	Použité	Dostupné	Využití čipu
Počet kombinačních bloků	502	12096	4 %
Počet klopných obvodů	185	6048	3 %
Počet blokových RAM pamětí	1	36	2 %

Pro srovnání lze uvést, že ve stejném obvodu jádro Spacewire CODEC od ESA zabírá více místa (viz **Tab. 4**) a minimální dosažitelná perioda systémového hodinového signálu je 15,4 ns (odpovídá kmitočtu 65 MHz) [5]. Je třeba ale poznamenat, že tento návrh má více funkcí.

Tab. 4: Využití zdrojů v RTAX 1000S – návrh Spacewire CODEC

	Použité	Dostupné	Využití čipu
Počet kombinačních bloků	739	12096	6 %
Počet klopných obvodů	419	6048	7 %

3.1.3 Implementace

Při implementaci návrhu je provedeno place-and-route (umístění buněk v FPGA a jejich propojení) tak, aby obvod byl funkční, což zahrnuje hlavně dodržení doby předstihu a přesahu. Problémem jsou asynchronní místa v návrhu, kde implementační nástroje mohou mít problémy. V tomto návrhu jsou to synchronizéry v přechodech mezi hodinovými doménami a vstupní část přijímače. K zajištění správné funkce těchto míst lze použít dodatečné příkazy. Pro nástroje Xilinx jsou to buď atributy ve VHDL kódu, nebo časová omezení uložená v UCF souboru.

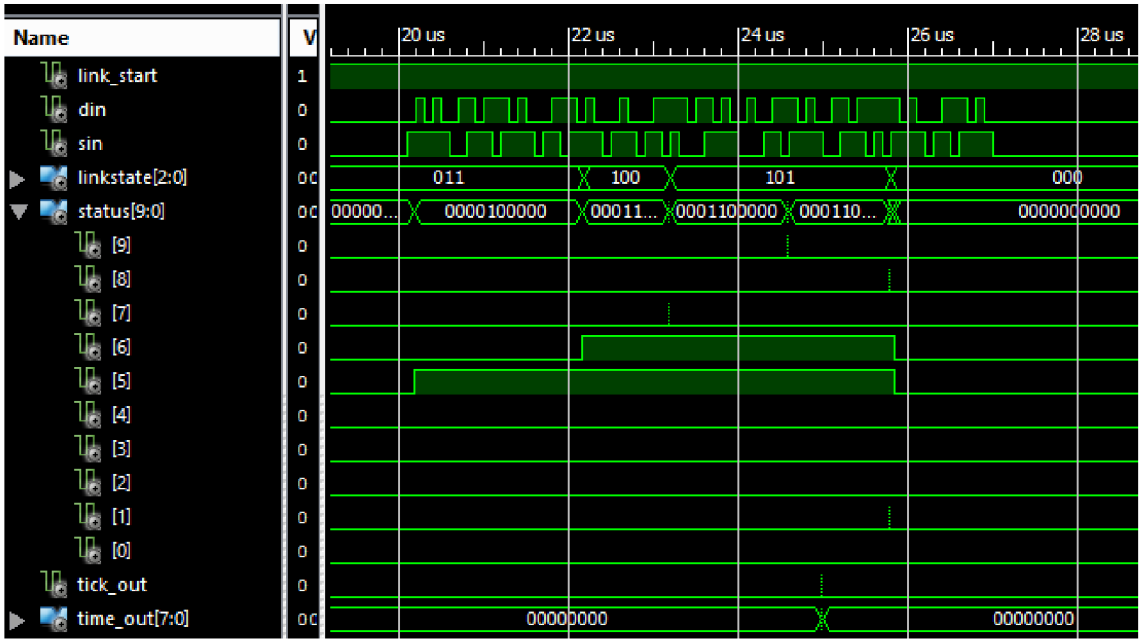
Synchronizéry v *RX Char* a *Kontrole rozpojení* jsou označeny atributem ASYNC_REG, který zajistí, aby oba klopné obvody byly co nejblíž k sobě. Dále se při jejich časování počítá, že na vstupu bude připojen asynchronní signál.

Na vstupních klopných obvodech v *RX Front* bylo třeba zajistit, aby byla dodržena doba předstihu při změně vstupu *Din*. Tato změna zároveň skrze hradlo XOR vytvoří

hranu na rekonstruovaném hodinovém signálu. Aby došlo ke správnému načtení vstupu do klopného obvodu, musí být vstup dostatečně dlouhou dobu ustálený. Toho bylo docíleno pomocí časového omezení OFFSET v UCF souboru. Ten způsobil zkrácení datové cesty a/nebo prodloužení cesty hodinového signálu, aby mezi zpožděními byl požadovaný rozdíl, v tomto případě 1 ns.

Po place-and-route byla provedena statická časová analýza, která určí časové parametry obvodu podle zpoždění použitých prvků a propojení mezi nimi. Požadavek na OFFSET byl splněn. Minimální dosažitelná perioda systémového hodinového signálu je 12,9 ns, což odpovídá kmitočtu 77,3 MHz.

Pro ověření funkčnosti vstupní části přijímače (*RX Front*) byla provedena post-route simulace (viz **Obr. 13**). Aplikována je na celý návrh, protože place-and-route musí proběhnout nad celým obvodem. Z důvodu úzkého zaměření simulace je proud vstupních dat poměrně krátký.



Obr. 13: Časový průběh některých signálů při post-route simulaci

Nejprve je vstup *link_start* nastaven do logické 1, což umožní inicializaci spoje. Po uplynutí obou zpoždění nutných pro přechod Řídící logiky do stavu *Started* (*linkstate* = „011“ – viz) je započato vysílání na vstupech *Din* a *Sin*. Reakce jádra na příchozí data je možné pozorovat na signálu *status* (pro vysvětlivky jednotlivých indexů viz **Tab. 1**). Nejprve je posláno několik náhodných bitů, *status(5)* indikuje přijetí prvního bitu. Další odezva je až na NULL kód – *status(6)*. To vyvolá také přechod do stavu *Connecting* (*linkstate* = „100“). Dalším znakem je FCT, jehož přijetí je signalizováno pulzem *status(7)*. Řídící logika přechází do stavu *Run* (*linkstate* = „101“). Následně je poslán

časový kód. Jeho přijetí je signalizováno pulzem *status(9)*. Po jeho zpracování je z přijímače předán aplikaci – skrz výstup *time_out* a indikační signál *tick_out*. Dalšími poslanými daty jsou NULL kód, EOP znak a na konci 2 NULL kódy. Protože však EOP znak má chybný paritní bit, je detekována paritní chyba – pulz signálu *status(1)*. Zároveň stavový automat přejde do stavu *ErorrReset* (*linkstate* = „000“) a přijímač je vypnut.

4 Závěr

Cílem této práce bylo popsat standard SpaceWire a na základě toho vytvořit návrh koncového bodu v síti SpaceWire v jazyce VHDL, který bude možné použít pro různé projekty, či jej použít jako základ pro implementaci pokročilejších funkcí.

V první části je rozebrán zmíněný standard a jeho součásti. Pozornost je věnována hlavně vrstvám týkající se HDL implementace koncového bodu – práce s příchozími a odchozími daty a komunikace se zařízením, jemuž rozhraní přísluší. Vynechána je fyzická vrstva, popisující kabely a konektory, a některé věci týkající se směrování a přepínačů.

Praktická část se zabývá samotným návrhem. Jádro bylo vytvořeno tak, aby splňovalo všechny požadavky dané standardem. Nejdůležitějšími a nejkomplexnějšími částmi jsou vysílač a přijímač. Ve vysílači bylo zvoleno použití jednoho hodinového signálu, což výrazně zjednodušuje logiku a odstraňuje problémy s převodem signálu mezi časovými doménami. Nevýhodou je rychlost omezená systémovým kmitočtem. Část starající se o serializaci znaku je oddělená od zbytku vysílače, a proto její nahrazení nebude vyžadovat změny ostatních částí. Přijímač příchozí data synchronizuje do systémové hodinové domény po jednotlivých znacích. To omezuje maximální rychlost přijímání na přibližně stejnou jako je systémový kmitočet. Protože obnovený hodinový signál z přijatých dat má poloviční frekvenci, je nutné zpracovávat data na obě hrany hodinového signálu. Aby nebylo nutné použít DDR klopné obvody, které je nutné ve většině FPGA skládat z jednoduchých klopných obvodů, jsou vstupní data převáděna na bitové páry, které je možné zpracovávat pouze na jednu hranu hodinového signálu. Díky tomu je potřeba méně klopných obvodů a časové požadavky nejsou s výjimkou vstupu tak striktní. U vstupu bylo třeba zajistit dodržení doby předstihu, kdy změna vstupu Data vyvolá zároveň hranu na hodinovém signálu. Pomocí časových omezení je proto třeba dostatečně zpozdit hodinový signál.

Příchozí data jsou ukládána do paměti FIFO, jejíž velikost je možné zvolit. Skrz její místo je pak řízen proud vstupních dat. Součástí vysílače není FIFO pro odesílání dat a v případě potřeby je nutné tuto paměť implementovat v nadřazené aplikaci. Na rozdíl od přijímacího však nepotřebuje téměř žádnou logiku navíc, a proto jeho implementace v případě, že by bylo potřeba, je velmi jednoduchá.

Správnost návrhu byla ověřena funkční verifikací a post-route simulací, ve které byly ověřeny časové parametry vstupní části přijímače, pracující s rekonstruovaným hodinovým signálem. Návrh předložený v této práci byl proveden s ohledem na efektivní implementaci z hlediska plochy na čipu. Pro ověření potřebných zdrojů v FPGA byla

provedena syntéza do dvou různých obvodů. Prvním z použitých obvodů bylo FPGA Spartan-3 XC3S200 od firmy Xilinx, ve kterém navržený IP Core zabírá asi 14 % dostupných zdrojů. Druhým cílovým obvodem bylo FPGA RTAX1000S od firmy Microsemi, certifikované pro lety do vesmíru. V tomto obvodu návrh zabírá jen asi 4 % místa.

Seznam použitých zdrojů

[1] PARKES, Steve. *SpaceWire User Guide* [online]. STAR-Dundee Limited, 2012 [cit. 2016-05-28]. ISBN 978-0-9573408-0-0. Dostupné z: https://www.star-dundee.com/sites/default/files/SpaceWire%20User's%20Guide.pdf

[2] *SpaceWire - Links, nodes, routers and networks* [online]. ECSS-E-ST-50-12C. Noordwijk, The Netherlands: ESA-ESTEC, 2003, 2008 [cit. 2016-05-28]. Dostupné z: http://www.ecss.nl/forums/ecss/dispatch.cgi/standards/docProfile/100654/d20090722143710/No/t100654.htm

[3] STAR-Dundee Ltd. *University of Dundee* [online]. [cit. 2015-12-13]. Dostupné z: http://www.dundee.ac.uk/main/business/casestudies/star-dundeeltd/

[4] ECSS-E-ST-50-5x series. *ESA SpaceWire* [online]. [cit. 2015-12-13]. Dostupné z: http://spacewire.esa.int/content/Standard/Draft_ECSS-E50-11.php

[5] SPACEWIRE-B. *ESA* [online]. [cit. 2015-12-13]. Dostupné z: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SpaceWire-b_-_HDL

[6] SPACEWIRE-RMAP. *ESA* [online]. [cit. 2015-12-13]. Dostupné z: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SpaceWire-RMAP_-_HDL

[7] SPACEWIRE-AMBA. *ESA* [online]. [cit. 2015-12-13]. Dostupné z: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SpaceWire-AMBA_-_HDL

[8] Advanced Microprocessor Bus Architecture (AMBA) Bus System. *Electronic design* [online]. [cit. 2015-12-13]. Dostupné z: http://electronicdesign.com/microprocessors/advanced-microprocessor-bus-architecture-amba-bus-system

[9] GRSPW2 SpaceWire Link. *Cobham Gaisler* [online]. [cit. 2015-12-13]. Dostupné z: http://www.gaisler.com/index.php/products?option=com_content

[10] GRSPW_CODEC SpaceWire Codec. *Cobham Gaisler* [online]. [cit. 2015-12-13]. Dostupné z: http://www.gaisler.com/index.php/products/ipcores/spacewire?task=view

[11] SpaceWire and SpaceFibre. *STAR-Dundee* [online]. [cit. 2015-12-13]. Dostupné z: https://www.star-dundee.com/knowledge-base/spacewire-and-spacefibre

[12] Interface Circuits for TIA/EIA-644 (LVDS). *Texas Instruments* [online]. [cit. 2015-12-13]. Dostupné z: http://www.ti.com/lit/an/slla038b/slla038b.pdf

[13] CHU, Pong P. *RTL hardware design using VHDL: coding for efficiency, portability, and scalability*. Hoboken, N.J.: Wiley-Interscience, 2006. ISBN 978-047-1720-928.