VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

VERIFIKACE KONCOVÉHO BODU V SÍTI SPACEWIRE

SPACEWIRE ENDPOINT VERIFICATION

BAKALÁŘSKÁ PRÁCE

BACHELOR´S THESIS

AUTOR PRÁCE — ONDŘEJ PEROUTKA

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: Ondřej Peroutka ***ID:*** 164358
Ročník: 3 ***Akademický rok:*** 2015/2016

NÁZEV TÉMATU:

Verifikace koncového bodu v síti SpaceWire

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte specifikaci sítě SpaceWire a navrhněte verifikační prostředí pro ověření funkčnosti koncového bodu sítě. Verifikační prostředí bude založeno na vrstvovém modelu dle metodiky VVM a popsáno v jazyce SystemVerilog. Nejnižší vrstva prostředí bude modelovat koncový bod sítě a také nadřazený systém pro ovládání IP coru koncového bodu sítě.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

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

Vedoucí práce: Ing. Vojtěch Dvořák
Konzultanti semestrální práce:

doc. Ing. Jiří Háze, Ph.D.
Předseda oborové rady

Abstrakt:

Tématem této bakalářské práce je verifikace SpaceWire IP coru vytvořeného na Ústavu mikroelektroniky Fakulty elektrotechniky a komunikačních technologií, VUT v Brně. Práce má 3 hlavní části. V první části práce je stručný popis standardu SpaceWire. Druhá část práce se zabývá teoretickým popisem verifikace. Poslední část práce se věnuje praktické části verifikace koncového bodu sítě SpaceWire.

Abstract:

The topic of the bachelor´s thesis is the verification of the SpaceWire endpoint IP core created at Department of Microelectronics, Faculty of Electrical Engineering and Communication, VUT Brno. The thesis has 3 major parts. The first part briefly describes the SpaceWire standard. The second part deals with the theoretical description of the verification. The last part deals with the verification of the SpaceWire endpoint.

Klíčová slova:

Funkční pokrytí, LVDS, SpaceWire, SystemVerilog, verifikace, verifikační prostředí

Keywords:

Functional coverage, LVDS, SpaceWire, SystemVerilog, verification, verification enviroment

Bibliografická citace

PEROUTKA, O. *Verifikace koncového bodu v síti SpaceWire*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 37 s. Vedoucí bakalářské práce Ing. Vojtěch Dvořák.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma **„Verifikace koncového bodu v síti SpaceWire“** 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 semestrální 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 této práce.

V Brně dne .....

.....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Úvod

SpaceWire je standard pro přenos dat, používaný především ve vesmírných družicích pro komunikaci mezi jednotlivými jednotkami. Připojenými jednotkami mohou být přístroje, senzory, procesory či velkokapacitní paměťové moduly. K propojení se používají obousměrné zdvojené datové linky pracující při rychlostech od 2 Mb/s do 200 Mb/s. Mezi hlavní výhody sítě SpaceWire patří rychlost, jednoduchost, nízká spotřeba a flexibilita [1].

Před standardem SpaceWire měla většina výrobců vybavení pro vesmírné družice svoje vlastní komunikační rozhraní. S narůstajícím počtem zařízení a měřících přístrojů v družicích se zvyšoval i počet použitých rozhraní vedoucí k vyšší ceně a delší době potřebné pro testování a sestavení. Proto bylo potřeba vyvinout jednotný standard pro komunikaci, který by vedl ke zjednodušení vývoje družic. Řešením měl být standard IEEE 1355-1995. Se standardem však bylo mnoho problémů. Evropská kosmická agentura (ESA) proto oslovila University of Dundee, aby tyto problémy prozkoumala a vyřešila. Pro zachování co největší kompatibility byly jako základ použity standardy IEEE 1355-1995 a ANSI/TIA/EIA-644. Výsledkem se stal standard SpaceWire, vydaný v lednu roku 2003. Dnes standard využívají agentury ESA, NASA, JAXA a další.

V této práci bude verifikován SpaceWire IP core vytvořený na Ústavu mikroelektroniky, který je odvozen od IP coru nabízeným agenturou ESA [2]. Verifikace bude založena na vrstvové metodice. Použitým jazykem bude SystemVerilog. Ve verifikačním prostředí bude také vytvořen model koncového bodu sítě, který bude komunikovat s testovaným IP corem, a nadřazený systém pro ovládání testovaného IP coru.

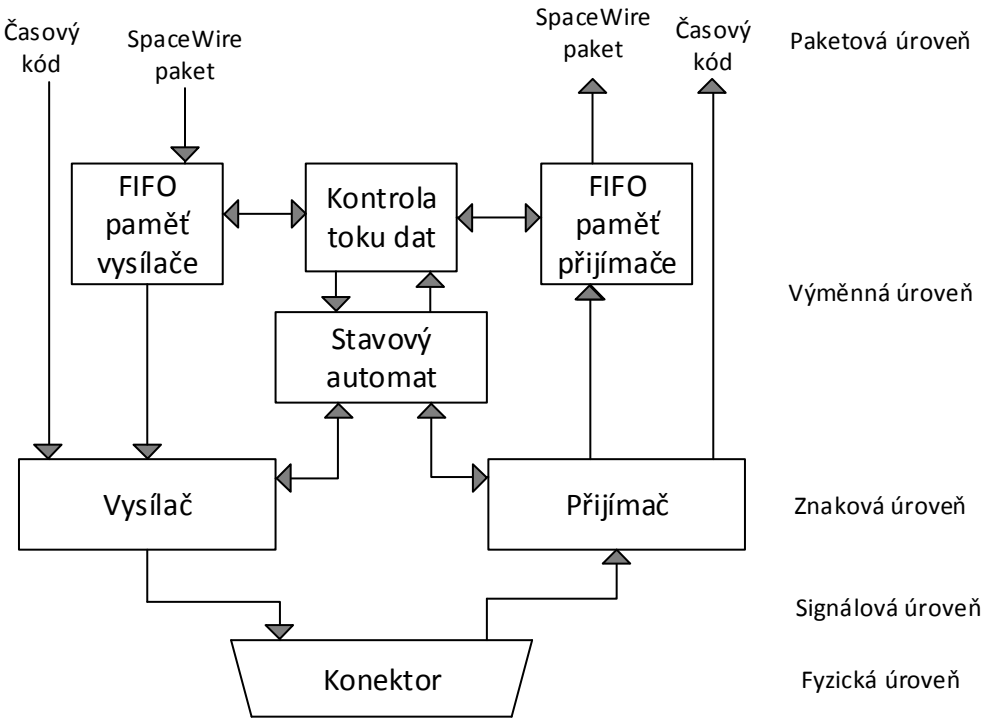
1 Popis standardu SpaceWire

Tato kapitola je zaměřena na popis standardu SpaceWire. Jednotlivé podkapitoly popisují úrovně SpaceWire definované ve standardu ECSS-E-ST-50-12C [3].

Standard SpaceWire je navržen jako vrstvový model s následujícími úrovněmi:

- **Fyzická úroveň**: Definuje konektory, kabely a jejich uspořádání, cesty na desce plošných spojů.
- **Signálová úroveň**: Definuje signálové zakódování, napěťové úrovně signálů, šumové tolerance, přenosové rychlosti.
- **Znaková úroveň**: Definuje datové a kontrolní znaky k ovládání toku dat skrz linku.
- **Výměnná úroveň**: Definuje postup inicializace linky, kontrolu toku dat, detekci chyb linky a obnovu linky z těchto chyb.
- **Paketová úroveň**: Definuje rozdělení přenášených dat do paketů.
- **Síťová úroveň**: Definuje strukturu SpaceWire sítě a způsob přenosu paketů z počátečního do cílového uzlu sítě.

Jednotlivé úrovně standardu SpaceWire jsou na obrázku 1-1.



Obrázek 1-1: Jednotlivé úrovně SpaceWire rozhraní

1.1 Fyzická úroveň

Fyzická úroveň zahrnuje kabely, konektory a vodivé cesty desek plošných spojů.

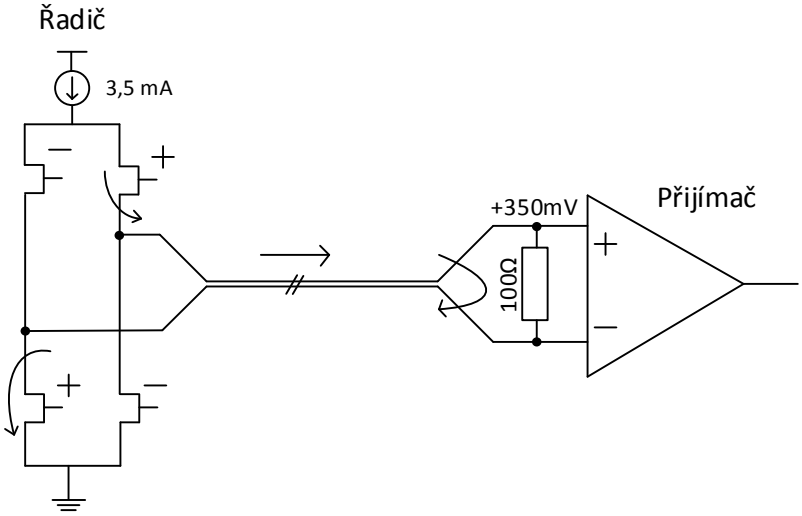
Přenos dat po lince probíhá pomocí dvou diferenčních signálových párů v obou směrech. Kabel pro SpaceWire tedy obsahuje čtyři kroucené dvojlinky. Pro dosažení rychlosti přenosu 200 Mb/s by neměla délka kabelu přesáhnout deset metrů. Kabel je zakončen konektorem D-typu s devíti kolíky, osm signálových kolíků a jeden pro stínění.

1.2 Signálová úroveň

Signálová úroveň zahrnuje signálové zakódování, napěťové úrovně signálů a šumové tolerance.

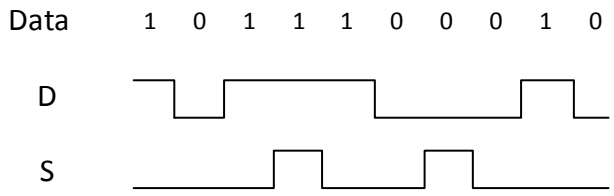
SpaceWire využívá LVDS (low voltage differential signalling) signalizaci a Data-Strobe enkódování.

LVDS je signalizace logických hodnot pomocí nízkonapěťových rozkmitů s typickou hodnotou 350 mV. U přijímače je zapojen zakončovací 100 ohmový rezistor, kterým prochází konstantní proud 3,5 mA, čímž na rezistoru vzniká úbytek napětí 350 mV. Použití této techniky zaručuje nízké hodnoty šumu a nízkou spotřebu i při vyšších rychlostech. Na obrázku 1-2 je zobrazena operace LVDS [4].



Obrázek 1-2: Operace LVDS

Data-Strobe (DS) kódování je způsob zakódování hodinového signálu a dat do signálů Data a Strobe. Hodinový signál získáme zpět použitím operace XOR na Data a Strobe. Signál Data jsou přímo hodnoty dat. Signál Strobe změní hodnotu, pokud dvě po sobě jdoucí hodnoty dat jsou stejné. Příklad DS kódování je na obrázku 1-3.

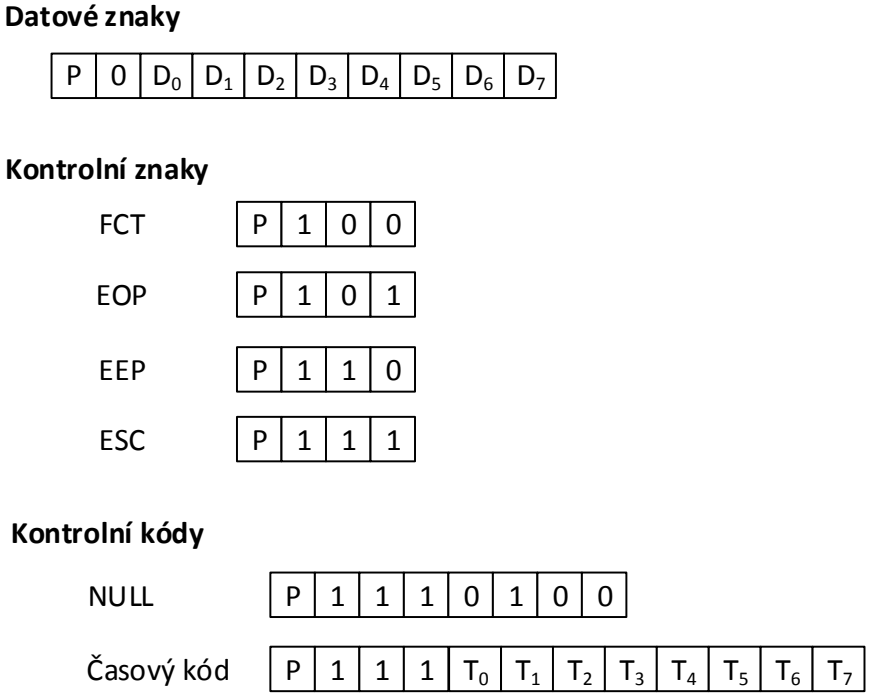


Obrázek 1-3: Data-Strobe kódování

Pokud není přijat další bit do 850 nanosekund, nastane rozpojení linky. Proto musí být minimální přenosová rychlost dat alespoň 2 Mb/s. Maximální rychlost přenosu dat není stanovena standardem, ale měla by být nastavena podle zkreslení a zpoždění signálu v konkrétní aplikaci. Poslední definovanou rychlostí je přenosová rychlost po restartu nebo rozpojení linky (10±1) Mb/s.

1.3 Znaková úroveň

Ve standardu SpaceWire jsou dva druhy znaků, datové a kontrolní. Datové znaky slouží k přenosu dat, kontrolní znaky zajišťují správnou komunikaci na lince a synchronizaci času mezi dvěma zařízeními. Dále jsou zde dva kontrolní kódy, NULL kód a časový kód. Jednotlivé znaky a kontrolní kódy jsou na obrázku 1-4.



Obrázek 1-4: Datové a kontrolní znaky a kontrolní kódy

Datové znaky jsou složeny z paritního bitu (P), kontrolní vlajky a osmi datových bitů (D_0 – D_7). Datové bity jsou řazeny od nejméně významného bitu (D_0) po nejvíce významný bit (D_7). Paritní bit zajišťuje lichou paritu pro datové bity z předchozího datového znaku nebo dva

bity předchozího kontrolního znaku, aktuální paritní bit a kontrolní vlajku. Pro datový znak je kontrolní vlajka v logické 0.

Kontrolní znaky mají také paritní bit a kontrolní vlajku (pro kontrolní znak v logické 1), ale místo datových bitů jsou zde dva kontrolní bity. Jsou možné čtyři kontrolní znaky:

- FCT (kontrola toku dat) – značí, že se mohou posílat datové znaky,
- EOP (řádný konec paketu) – paket byl úspěšně odeslán a přijat,
- EEP (chybný konec paketu) – část paketu je chybná,
- ESC (únikový znak) – používá se v kontrolních kódech.

FCT

Znak pro kontrolu toku dat slouží k inicializaci linky a kontrole přenosu dat. Při inicializaci linky je znak FCT posílán po přijetí kódu NULL k dokončení inicializace linky. Při normální operaci linky odeslání FCT znaku indikuje, že ve FIFO paměti přijímače je místo pro osm dalších datových znaků nebo znaků pro konec paketu. Po přijetí FCT znaku druhým koncem linky je vysílači povoleno poslat až osm dalších znaků. Pokud je ve FIFO paměti více místa, je odeslán jeden FCT znak za každé místo pro osm znaků. Například pokud je odesláno pět FCT znaků, v paměti je volno pro alespoň čtyřicet znaků.

EOP

Znak pro řádný konec paketu je odeslán, pouze pokud je celý paket řádně odeslán a během posílání nenastala žádná chyba. Při chybě jsou všechna data až do dalšího znaku pro řádný konec paketu zahozena.

EEP

Znak pro chybný konec paketu značí, že během posílání paketu došlo k chybě a paket není kompletní. Pokud dojde k chybě na lince během posílání paketu, do FIFO paměti přijímače na obou koncích se vloží znak pro chybný konec paketu, jakmile je v paměti dostatek místa. Takto ukončený paket pokračuje dále ve SpaceWire síti. Zbytek neodeslaného paketu je zahozen.

Kontrolní kód NULL je složen ze znaku ESC následován znakem FCT. Pokud na lince nejsou posílána data, je posílán NULL. Časový kód se skládá ze znaku ESC následován jedním datovým znakem. Časový kód slouží k rozesílání času pro ostatní uzly v síti nebo jejich synchronizaci. Časový údaj je uložen v šesti nejméně významných datových bitech datového znaku ($T_0 – T_5$). Zbylé dva datové bity jsou kontrolní vlajky ($T_6 – T_7$).

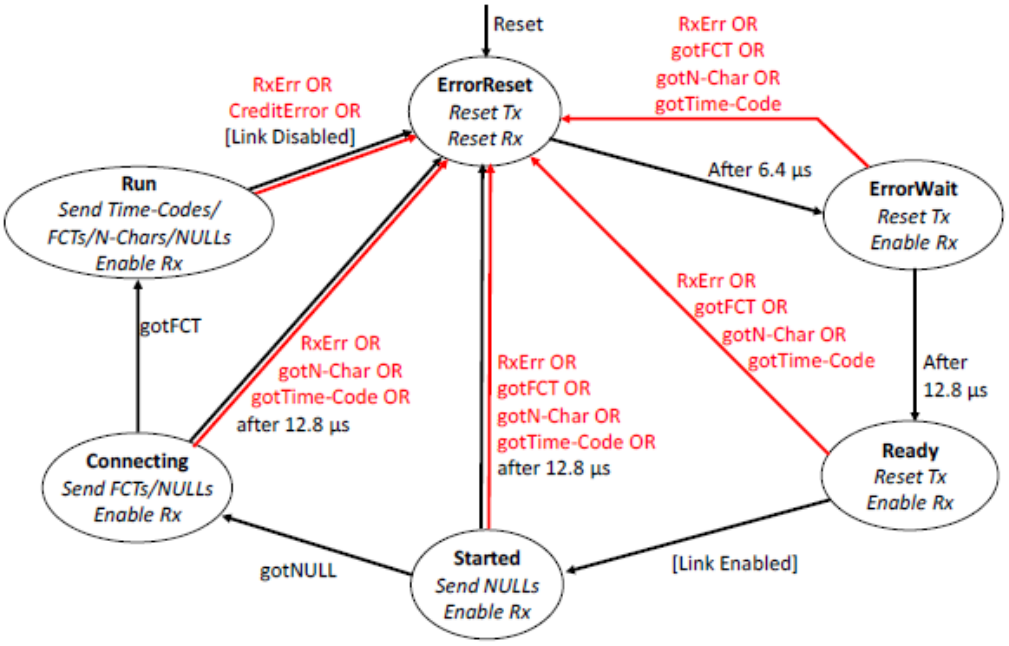
Při odesílání mají znaky následující prioritu:

1. Časový kód
2. FCT
3. Datové znaky, EOP a EEP
4. NULL

1.4 Výměnná úroveň

Výměnná úroveň definuje výměnu dat mezi dvěma konci linky. Je zde popsán postup inicializace linky, kontrola toku dat po lince a postup obnovy linky z chyb.

Pakety k odeslání jsou uloženy ve FIFO paměti vysílače. Přijaté pakety jsou uloženy do FIFO paměti přijímače, odkud je může číst nadřazený systém. Velikost dat v paměti je řízena pomocí znaku pro kontrolu toku dat. Odeslání znaku pro kontrolu toku dat značí, že v paměti je místo pro osm dalších znaků. Časové kódy k odeslání jsou zařazeny za právě posílaný znak. Stavový automat kontroluje inicializaci linky, detekuje chyby a řídí případnou obnovu z chyb inicializací linky. Stavový diagram obnovy linky je na obrázku 1-5, červeně jsou zobrazeny možné chyby při obnově linky.



Obrázek 1-5: Stavový diagram pro obnovu linky [1]

Linka je inicializována po synchronizaci obou konců linky. Bitová synchronizace je provedena dekódováním data-strobe signálů, znaková synchronizace je provedena během inicializace linky.

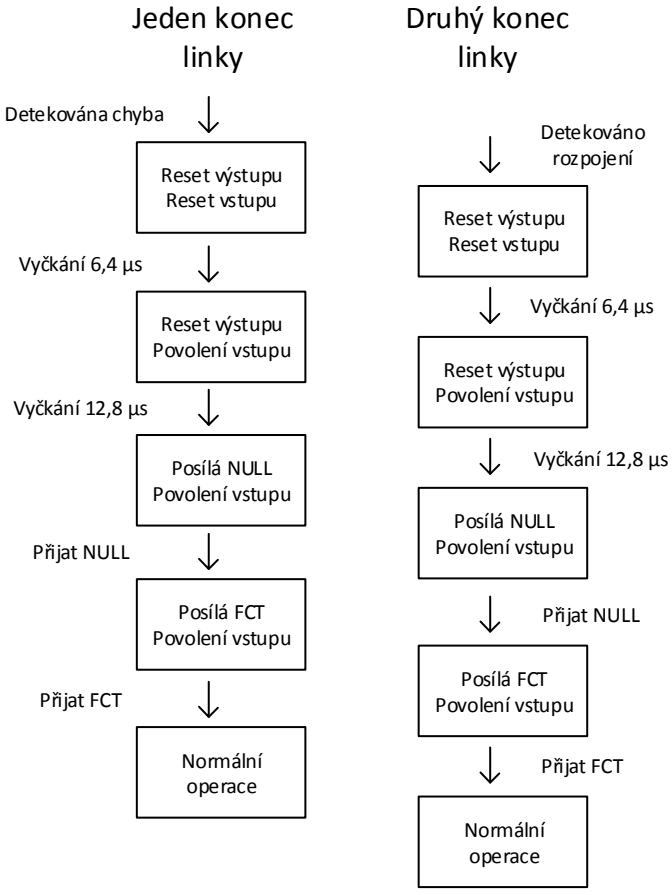
Inicializace linky začíná posláním kontrolního kódu NULL z jednoho uzlu, který poté čeká na přijetí NULL kódu od opačného uzlu. Jakmile je NULL kód přijat, je odeslán FCT znak a čeká se na jeho přijetí. Jakmile oba uzly přijmou FCT, je linka plně inicializována.

Na lince mohou nastat tyto chyby:

- **Chyba rozpojením**: Linka je rozpojena, takže jeden nebo oba uzly nedostávají data nebo NULL kód. Rozpojení nastane, pokud nejsou přijaty nové znaky do 850 nanosekund.

- **Chyba parity**: Znak obsahuje chybu parity.
- **Úniková chyba:** Únikový znak je následován dalším únikovým znakem, znakem pro konec paketu nebo znakem pro chybný konec paketu, což jsou nepovolené sekvence.
- **Kreditová chyba:** Do FIFO paměti přijímače dorazí další datový znak, aniž by pro něj bylo místo.

Při detekci chyby přejde jeden konec linky do reset stavu a přestane komunikovat. Po vyčkání doby 6,4 μs. Protože druhý konec linky nedostal další znak po dobu delší než 850 ns, detekuje chybu rozpojením a přechází také do reset stavu. Druhý konec také čeká 6,4 μs. Poté oba konce vyčkávají dalších 12,8 μs, aby byly oba konce připraveny přijímat znaky. Následuje inicializace linky vzájemnou výměnou NULL kódů a FCT znaků. Oba konce posílají kódy NULL a čekají na přijetí kódu NULL. Po přijetí kódu NULL posílají znaky FCT. Když konec linky přijme znak FCT, přechází do stavu normální operace. Proces restartu linky je na obrázku 1-6.



Obrázek 1-6: Proces restartu linky

1.5 Paketová úroveň

Paketová úroveň definuje části SpaceWire paketu:

- cílovou adresu,
- náklad,
- znak pro konec paketu (EOP).

Jednotlivé části SpaceWire paketu jsou na obrázku 1-7.

Obrázek 1-7: SpaceWire paket

Jako první se odesílá cílová adresa. Je to seznam datových znaků označující identitu cílového uzlu, nebo jakou cestou musí paket projít skrz sít k cílovému uzlu. V případě přímého propojení odesílajícího a cílového uzlu není adresa potřeba. Ve druhé části paketu jsou přepravovaná data. Tato část může být libovolně velká. Jako poslední se odešle znak pro konec paketu. Jakýkoliv další datový znak se považuje za začátek dalšího paketu.

V případě, že se během posílání paketu vyskytne chyba, ukončí se paket znakem pro chybný konec paketu. Zbytek neodeslaného paketu je zahozen.

1.6 Síťová úroveň

Tato vrstva definuje SpaceWire rozbočovače, uzly a sítě. Dále jsou popsány možné chyby v síti a je definován protokol obnovy na síťové úrovni.

Sít je tvořena linkami, uzly a rozbočovači. Linky zajišťují přenos paketů mezi jednotlivými uzly sítě. Uzly jsou počátkem a cílem paketů. Jako uzel může být procesor či senzor. Napřímo může být propojeno pouze omezené množství uzlů. Pro propojení více uzlů se používají rozbočovače.

K dopravení do cílového uzlu potřebuje mít paket jeho adresu. Adresování paketů se provádí dvěma způsoby. V prvním způsobu mají jednotlivé uzly přiřazený index s logickou adresou. Paket má pak jako cílovou adresu právě tento index. Rozbočovače mají nadefinovanou tabulku s indexy a jim odpovídající porty, a podle toho směrují příchozí pakety. Ve druhém případě je cílovou adresou seznam indexů portů, kterými se paket dostane do cílového uzlu.

Metodou pro směrování paketů je ve SpaceWire síti směrování červí dírou. Jakmile rozbočovač přečte adresu paketu, určí, kterým portem paket pošle dál. Pokud je port volný, ihned směruje paket k tomuto portu, aniž by čekal na celý paket, a tento port zablokuje. Port je uvolněn, jakmile je odeslán znak pro konec paketu. Nevýhodou je dlouhé blokování portů, pokud prochází dlouhý paket. To může vést až k řetězové blokaci po síti.

2 Verifikace

Tato kapitola se věnuje obecnému popisu verifikace a krátkému představení jazyka SystemVerilog.

Proces verifikace slouží k ověření funkčnosti navrhovaného obvodu a nalezení případných chyb oproti zadaným požadavkům. Verifikace se provádí simulací ověřovaného obvodu. Nejčastěji používaným jazykem pro verifikaci je SystemVerilog.

2.1 Obecný popis verifikace

Verifikace se řídí verifikačním plánem. Na základě zadaných specifikací se přesně definuje, jaké funkce obvodu se budou verifikovat a jakým způsobem. Kvalita verifikačního plánu se dá určit z pokrytí kódu při testování. To znamená, jak velká část kódu byla při verifikaci otestována. Kritérii mohou být provedení všech řádků kódu, použití jednotlivých větví v příkazech if – else, přiřazení všech možných pravdivostních hodnot ve výrazech nebo jestli prošel stavový automat všemi stavy a možnými přechody. Existuje ještě funkční pokrytí. Funkční pokrytí znamená, které funkce obvodu vypsané ve verifikačním plánu byly provedeny testovaným designem [5].

2.1.1 Body funkčního pokrytí

Body funkčního pokrytí jsou dány požadavky na funkce testovaného designu, které by měl design při verifikaci splnit. Seznam těchto funkcí je uveden ve verifikačním plánu spolu s postupem, jak budou tyto funkce ověřeny. Splnění všech bodů funkčního pokrytí při verifikaci znamená 100% funkční pokrytí a verifikovaný design splňuje všechny vznesené požadavky na jeho činnost.

2.1.2 Generování vstupních dat pro testovaný design

Vstupní data můžeme generovat několika způsoby:

- Přímými testy
- Testy s náhodnými čísly
- Verifikací řízenou pokrytím

U přímých testů jsou data pro testovaný design přímo vypsána testovacím inženýrem. Výhodou je krátká doba potřebná pro vytvoření testu. Nevýhody jsou nutnost velkého počtu testů u složitějších designů, potřeba upravení všech testů při změně specifikací a testování pouze očekávaného chování obvodu. Z těchto důvodů se přímé testy používají pro jednoduché designy nebo otestování funkcí, které nepokryl test s náhodnými čísly.

Testy s náhodnými čísly mají data generována funkcí *random*. Pro rychlejší dosažení plného funkčního pokrytí jsou data generována z určitého intervalu nebo s určitou

pravděpodobností. Oproti přímým testům je dosáhnutí 100% pokrytí testy s náhodnými čísly rychlejší a otestuje se i nepředpokládané chování obvodu. Naopak vytvoření verifikačního prostředí zabere více času a některé funkce se nemusí podařit otestovat. Na tyto funkce se poté musí zaměřit přímé testy.

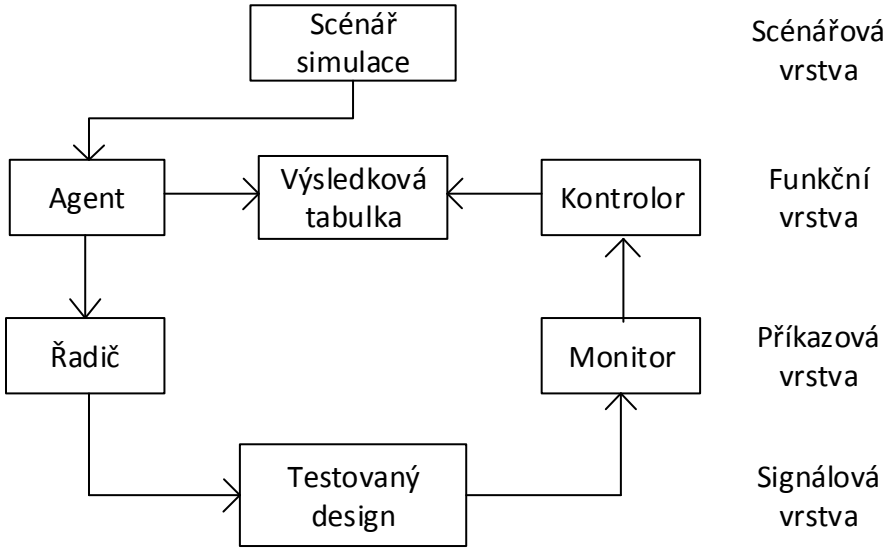
Při verifikace řízenou pokrytím je snaha o vyrovnané funkční pokrytí a pokrytí kódu. Vysoká hodnota funkčního pokrytí ale malá hodnota pokrytí kódu může například znamenat, že funkční model neobsahuje všechny důležité funkce testovaného designu. Naopak vysoká hodnota pokrytí kódu a malá hodnota funkčního pokrytí značí například neúplnost designu. Tato metoda je dnes nejprosazovanější.

2.1.3 Vrstvový model verifikačního prostředí

Vrstvové verifikační prostředí je složeno z několika bloků na různých vrstvách. To umožňuje snadnější úpravy verifikačního prostředí při změnách specifikace, snazší opravu případných chyb a možnost použití bloků při verifikaci jiného obvodu. Vrstvové verifikační prostředí má tyto vrstvy:

- Signálová
- Příkazová
- Funkční
- Scénářová

Příklad vrstvového verifikačního prostředí je na obrázku 2-1. V případě testování jednoduchých designů jsou některé bloky sloučeny do jednoho. Pokud je testovaný design složitější, mohou být některé bloky ve verifikačním prostředí i vícekrát.



Obrázek 2-1: Příklad vrstvového verifikačního prostředí

Signálová vrstva

Signálová vrstva je nejnižší vrstva verifikačního prostředí obsahující pouze testovaný design. Logické hodnoty vstupních signálů pro testovaný design jsou řízeny driverem podle příkazů od agenta. Logické hodnoty výstupních signálů jsou zaznamenávány a kontrolovány monitorem.

Příkazová vrstva

V příkazové vrstvě jsou řadič a monitor.

Řadič převádí příkazy od agenty na změny logických hodnot signálů přivedených do testovaného designu.

Monitor provádí kontrolu dat na úrovni výstupních signálů testovaného obvodu a převádí získané hodnoty na příznak pro kontrolora. Dalšími možnostmi kontroly dat jsou porovnání s referenčním modelem, což je abstraktní popis obvodu generující stejné výstupy jako testovaný design, nebo pomocí assertions (tvrzení) v RTL.

Funkční vrstva

Funkční vrstva obsahuje agenta, kontrolora a výsledkovou tabulku.

Agent převádí příkazy z vyšší vrstvy na příkazy pro driver a posílá testovací vektory do výsledkové tabulky.

Výsledková tabulka sleduje funkční pokrytí simulace a porovnává data od agenta s daty od kontrolora.

Kontrolor přijímá příznaky od monitoru z nižší vrstvy a převádí je na data pro výsledkovou tabulku.

Scénářová vrstva

Scénářová vrstva je nejvyšší vrstva verifikačního prostředí obsahují scénář simulace, ve kterém jsou zapsány příkazy pořadí, tak jak mají být provedeny.

2.1.4 Verifikační plán

Verifikační plán je dokument, kterým se řídí proces verifikace obvodu. Plán obsahuje seznam bodů funkčního pokrytí, které testovaný design musí splnit. Body pokrytí jsou sestaveny podle specifikace a požadavků na testovaný obvod, například odesílání znaků podle jejich priority. U každého bodu pokrytí je i způsob jeho ověření, například přímými testy. Cílem verifikačního plánu je plné funkční pokrytí a pokrytí kódu. V takovém případě provedl testovaný obvod všechny řádky svého RTL kódu (register-transfer level – obsahuje abstraktní popis modelu obvodu) a splnil veškeré funkční požadavky.

2.2 SystemVerilog

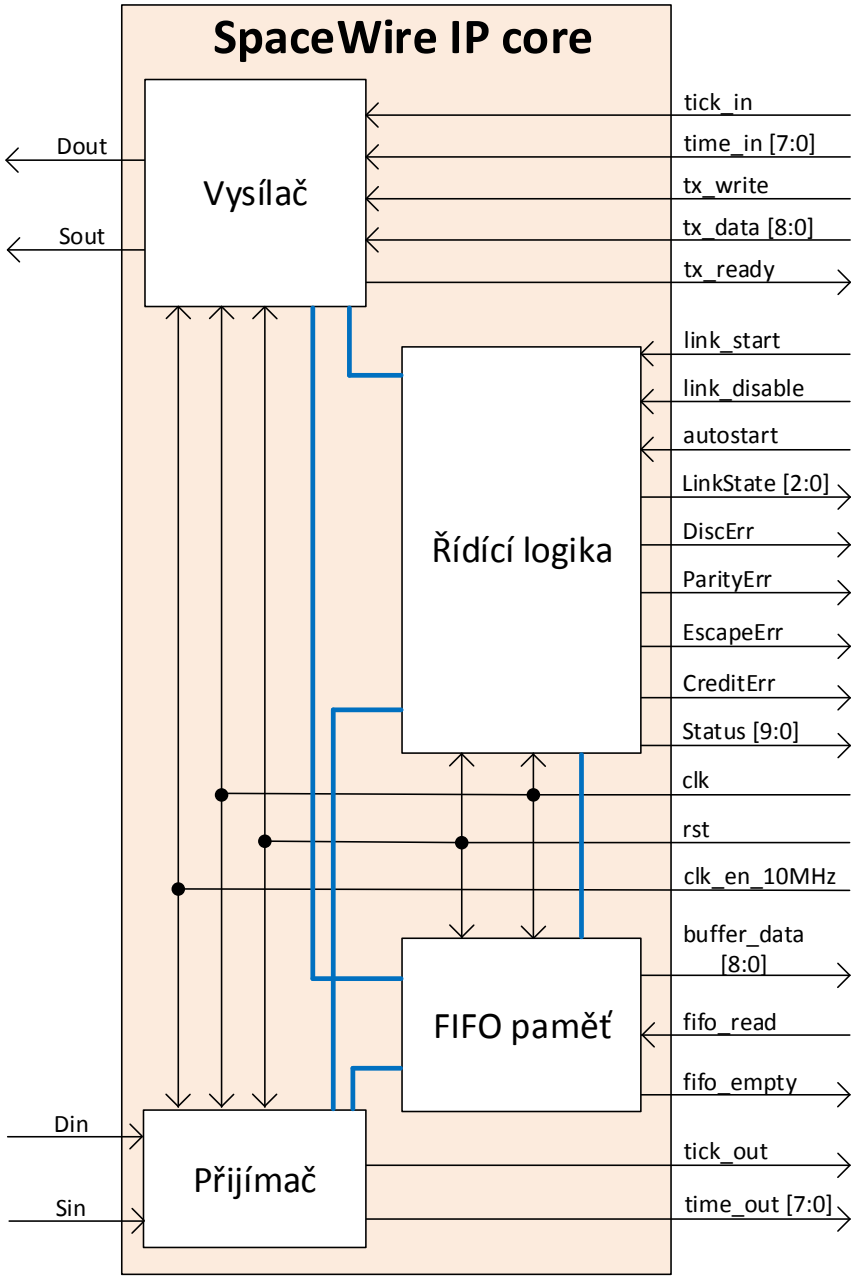
SystemVerilog je vyšší programovací jazyk digitálních obvodů, umožňující jejich popis i verifikaci. SystemVerilog je popsán ve standardu IEEE 1800-2012. Za vznikem stojí společnost Accellera. SystemVerilog je rozšířením jazyka Verilog 2001. Byly rozšířeny a přidány nové konstrukce pro dosažení vyšší úrovně abstrakce pro modelování a verifikaci obvodů, zjednodušení verifikace složitějších obvodů a možnost znovupoužití částí verifikačního kódu [6].

3 Testovaný SpaceWire design a verifikační plán

Tato kapitola se věnuje představení verifikovaného designu vytvořeného na Ústavu mikroelektroniky. Dále je zde představen verifikační plán pro ověření funkčnosti tohoto designu.

3.1 Testovaný SpaceWire design

Tato podkapitola se věnuje popisu vnějších signálů testovaného designu, se kterými se bude pracovat ve verifikačním prostředí, a způsobu komunikace s testovaným designem. Jednotlivé signály je možno vidět na obrázku 3-1.



Obrázek 3-1: Vnější signály designu (černá) a schematické propojení vnitřních bloků (modrá)

Nadřazená aplikace zajišťuje designu systémový hodinový signál, povolovací signál s frekvencí 10 MHz a signál pro reset designu. Dále aplikace ovládá komunikaci po lince pomocí signálů *link_start, link_disable* a *autostart. Link_start* povoluje inicializaci linky, *link_disable* naopak slouží k rozpojení linky. Pokud je *autostart* v logické 1, design vyčkává ve stavu *Ready* a s inicializací linky čeká, dokud z opačného konce nepřijde první NULL kód.

Jakmile je design připraven načíst datový znak nebo časový kód k odeslání a druhý konec linky si vyžádal N-znaky (data, EOP, EEP), nastaví signál *tx_ready* do logické 1. Pokud má aplikace připraven časový kód, nastaví signál *tick_in* do logické 1 a design načte hodnotu signálu *time_in*. Pokud je připraven datový znak, tak aplikace nastaví logickou 1 na signálu *tx_write,* design načte hodnotu signálu *tx_data* a zkontroluje hodnotu nejnižšího bitu. Pro logickou hodnotu 0 odešle design datový znak s hodnotou nejvyšších osmi bitů. Pokud je nejnižší bit v logické 1, pošle design znak pro konec paketu podle logické hodnoty druhého nejnižšího bitu: EOP pro log. hodnotu 0 nebo EEP pro log. hodnotu 1.

Všechny znaky k odeslání a hodinový signál jsou zakódovány pomocí Data-Strobe kódování a posílány signály *Dout* a *Sout* na druhý konec linky. Data z druhého konce linky jsou přijímána skrz signály *Din* a *Sin,* následně rozkódována a uložena do FIFO paměti přijímače.

Jakmile jsou ve FIFO paměti uložené znaky, změní design logickou hodnotu signálu *fifo_empty* do 0 a data jsou připravena na výstupu *buffer_data.* Stejně jako u vstupních dat i zde nejméně významný bit rozhoduje, zda se jedná o datový znak (logická 0) nebo znak pro konec paketu (logická 1). Aplikace při vyčtení dat změní hodnotu signálu *fifo_read* na logickou 1 a design postupně vystavuje data uložena ve FIFO paměti. Pokud je přijat časový kód, design tento kód neukládá do paměti, ale hned odesílá na výstup *time_out* spolu s indikací o přijetí časového kódu nastavením signálu *tick_out* do logické 1.

Signály *Link_State, DiscErr, ParityErr, EscapeErr, CreditErr* a *Status* slouží aplikaci k monitorování stavu linky a případných chyb při komunikaci. Signálem *Link_State* indikuje design svůj aktuální stav. *DiscErr, ParityErr, EscapeErr, CreditErr* indikují chyby ve stavu *Run. DiscErr* značí uplynutí doby 850 nanosekund od přijetí posledního bitu, *ParityErr* značí paritní chybu v přijatých znacích, *EscapeErr* je přijetí nepovolené kombinace znaků a *CreditErr* značí přijetí více znaků, než kolik jich bylo vyžádáno. Signál *Status* informuje nadřazenou aplikaci o typu přijatých znaků a nastalé chyby v jakémkoliv stavu.

Souhrn všech signálu spolu s jejich směrem vzhledem k designu, počtem bitů a funkci je v tabulce 3-1.

Tabulka 3-1: Přehled vnějších signálů testovaného designu

Název signálu	Směr signálu	Počet bitů	Aktivní hodnota	Funkce signálu
rst	vstup	1	'1'	Reset designu
clk	vstup	1	náběžná hrana	Systémový hodinový signál
clk_en_10MHz	vstup	1	'1'	Povolovací signál s frekvencí 10 MHz
link_start	vstup	1	'1'	Příkaz k inicializaci linky
link_disable	vstup	1	'1'	Příkaz k rozpojení linky
autostart	vstup	1	'1'	Povolení automatické inicializace linky po přijetí kódu NULL
Link_State	vstup	3	-	Aktuální stav stavového automatu
tick_in	vstup	1	'1'	Příkaz k odeslání časového kódu
time_in	vstup	8	-	Bity časového kódu k odeslání
tx_write	vstup	1	'1'	Aplikace zapisuje data k odeslání
tx_data	vstup	9	-	Datové bity k odeslání. Nejnižší bit určuje datový znak nebo znak pro konec paketu
fifo_read	vstup	1	'1'	Jsou vyčítána data z FIFO paměti
fifo_empty	výstup	1	'0'	Ve FIFO paměti jsou uložena data k vyčtení
tick_out	výstup	1	'1'	Časový kód připraven na výstupu
time_out	výstup	8	-	Bity přijatého časového kódu
tx_ready	výstup	1	'1'	Připraven k načtení dalších dat k odeslání
buffer_data	výstup	9	-	Přijaté bity datového znaku
DiscErr	výstup	1	'1'	Chyba rozpojením linky ve stavu Run
ParityErr	výstup	1	'1'	Chyba parity ve stavu Run
EscapeErr	výstup	1	'1'	Přijetí nepovolené kombinace znaků ve stavu Run
CreditErr	výstup	1	'1'	Přijetí nevyžádaných datových znaků ve stavu Run
Status[0]	výstup	1	'1'	Chyba rozpojením linky
Status[1]	výstup	1	'1'	Chyba parity
Status[2]	výstup	1	'1'	Přijetí nepovolené kombinace znaků
Status[3]	výstup	1	'1'	Přijetí nevyžádaných datových znaků
Status[4]	výstup	1	'1'	Druhý konec linky si vyžádal více než 56 datových znaků najednou
Status[5]	výstup	1	'1'	Detekování přijatí prvního bitu
Status[6]	výstup	1	'1'	Přijetí NULL kódu
Status[7]	výstup	1	'1'	Přijetí FCT znaku
Status[8]	výstup	1	'1'	Přijetí datového znaku nebo znaku pro konec paketu
Status[9]	výstup	1	'1'	Přijetí časového kódu

3.2 Verifikační plán

V této podkapitole jsou rozepsány jednotlivé testy pro verifikaci spolu s jejich postupem. Verifikována bude pouze funkční část designu. Při přechodu mezi jednotlivými testy bude testovaný design i BFM model (bus functional model, dále jen BFM model, viz kapitola 4) vyresetován. Pokud těmito testy nebude dosáhnuto 100% pokrytí kódu, budou dopsány další testy pro dosažení 100% pokrytí kódu.

3.2.1 Test inicializace linky s link_start a link_disable

Tento test bude verifikovat schopnost designu inicializovat a rozpojit linku na příkazy od nadřazeného systému pomocí signálů *link_start* a *link_disable*.

Průběh testu

Nejprve bude povoleno generování hodinových signálů a provedeno vyresetování designu. Poté bude zavolána procedura na nastavení signálu *link_start* do logické 1 a spuštění inicializace linky. Vyčká se na stav *Run* u testovaného designu i BFM modelu. V tomto stavu se poté bude čekat po dobu 200 nanosekund. Signál *link_disable* bude nastaven do logické 1 a bude se čekat po dobu 20 mikrosekund. Design by se měl dostat přes stavy *Error_Reset* a *Error_Wait* do stavu *Ready*, což zabere čas 19,2 mikrosekundy, a v tomto stavu by měl setrvat, protože *link_disable* je v logické 1.

Body funkčního pokrytí

Během tohoto testu budou sledovány tyto přechody na signálu *LinkState* (0 = Error_Reset, 1 = Error_Wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run):

1. (0 => 1), (1 => 2), (2 => 3), (3 => 4), (4 => 5) když je *link_start* v logické 1.
2. (0 => 1), (1 => 2) když je *link_start* a *link_disable* v logické 1.

3.2.2 Test inicializace linky s autostart

Tento test verifikuje schopnost designu automaticky inicializovat linku na příkaz signálem *autostart* a obdržení kódu NULL.

Průběh testu

Na začátku testu bude nastaven signál *autostart* do logické 1. Vytvořený BFM model poté začne navazovat s testovaným designem spojení. Vyčká se na stav *Run* u testovaného designu i BFM modelu. V tomto stavu se poté bude čekat po dobu 200 nanosekund. Signál *link_disable* bude nastaven do logické 1 a bude se čekat po dobu 20 mikrosekund. Design by se měl opět dostat do stavu *Ready* a v tomto stavu setrvat.

Body funkčního pokrytí

Během tohoto testu budou sledovány tyto přechody na signálu *LinkState* (0 = Error_Reset, 1 = Error_Wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run):

1. (0 => 1), (1 => 2), (2 => 3), (3 => 4), (4 => 5) když je *autostart* v logické 1.
2. (0 => 1), (1 => 2) když je *autostart* a *link_disable* v logické 1.

3.2.3 Test restartu linky podle stavového diagramu pro obnovu linky

V tomto testu se budou verifikovat přechody stavů podle stavového diagramu (viz Obrázek 1-5).

Průběh testu

Testovaný design bude postupně uveden do všech stavů (kromě stavu *Run*) a navozeny všechny možné chyby v těchto stavech, které způsobí přechod do stavu *Error_Reset*. BFM model bude držen ve stavu, kdy neposílá znaky. Jednotlivé chyby budou navozeny voláním procedur pro odeslání znaků v definovaném pořadí, které tyto chyby způsobí.

Přechody stavů

Do stavu *Error_Wait* ze stavu *Error_Reset* se design dostane po uplynutí 6,4 µs. Přechod *Error_Wait* => *Error_Reset* je způsoben rozpojením linky, paritní chybou, únikovou chybou, přijetím FCT znaku, přijetím datového znaku nebo znaků pro konec paketu, přijetím časového kódu.

Do stavu *Ready* ze stavu *Error_Reset* se design dostane nejprve přechodem *Error_Reset* => *Error_Wait* po uplynutí času 6,4 µs a následně přechodem *Error_Wait* => *Ready* po uplynutí času 12,8 µs. Přechod *Ready* => *Error_Reset* je způsoben rozpojením linky, paritní chybou, únikovou chybou, přijetím FCT znaku, přijetím datového znaku nebo znaků pro konec paketu, přijetím časového kódu.

Do stavu *Started* ze stavu *Error_Reset* se design dostane nejprve přechodem *Error_Reset* => *Error_Wait* po uplynutí času 6,4 µs, přechodem *Error_Wait* => *Ready* po uplynutí času 12,8 µs a následně přechodem *Ready* => *Started* pokud je *link_start* v logické 1 nebo *autostart* v logické 1 a je přijat kontrolní kód NULL. V tomto testu bude přechod *Ready* => *Started* povolen nastavení signálu *link_start* do logické 1. Přechod *Started* => *Error_Reset* je způsoben rozpojením linky, paritní chybou, únikovou chybou, přijetím FCT znaku, přijetím datového znaku nebo znaků pro konec paketu, přijetím časového kódu, uplynutí doby 12,8 µs bez přijetí kódu NULL.

Do stavu *Connecting* ze stavu *Error_Reset* se design dostane nejprve přechodem *Error_Reset* => *Error_Wait* po uplynutí času 6,4 µs, přechodem *Error_Wait* => *Ready*

po uplynutí času 12,8 µs, přechodem *Ready => Started* pokud je *link_start* v logické 1 nebo *autostart* v logické 1 a je přijat kontrolní kód NULL, a nakonec přechod *Started => Connecting* přijetím kódu NULL. V tomto testu bude přechod *Ready => Started* povolen nastavení signálu *link_start* do logické 1. Přechod *Started => Error_Reset* je způsoben rozpojením linky, paritní chybou, únikovou chybou, přijetím FCT znaku, přijetím datového znaku nebo znaků pro konec paketu, přijetím časového kódu, uplynutí doby 12,8 µs bez přijetí FCT znaku.

Kombinace znaků pro navození jednotlivých chyb

Každá sekvence odesílaných znaků začne povolením generování hodinové události (nahrazení hodinového signálu pro zmenšení nároků na simulaci) a následné volání procedur pro odeslání znaků. Rychlost odesílání bitů je pro tento test nastavena na frekvenci 10 MHz. Po ukončení vysílání znaků budou všechny použité proměnné a signály nastaveny do jejich původní hodnoty.

Detekce rozpojení linky je povoleno po přijetí prvního bitu. Proto bude odeslán jeden znak ESC následovaný přerušením komunikace po dobu 1,5 µs. Detekce ostatních chyb a znaků je povolena po přijetí prvního kódu NULL, proto u všech následujících chyb budou nejprve odeslány 2 kódy NULL. Paritní chyba bude navozena odesláním znaku ESC s nesprávným paritním bitem. Úniková chyba bude navozena odesláním dvou po sobě jdoucích znacích ESC. U chyby přijetím znaku FCT bude odeslán FCT znak následovaný kódem NULL. U chyby přijetím datového znaku bude odeslán datový znak následovaný kódem NULL. Chyba přijetí časového kódu bude navozena odesláním časového kódu následovaného kódem NULL. Chyba uplynutí 12,8 µs bude navozena ve stavu *Started* neodesláním kódu NULL, linka bude udržována aktivní odesíláním znaku ESC, a ve stavu *Connecting* neodesláním znaku FCT, linka bude udržována aktivní odesíláním kódu NULL.

Body funkčního pokrytí

Během tohoto testu budou sledovány tyto přechody na signálu *LinkState* (0 = *Error_Reset*, 1 = *Error_Wait*, 2 = *Ready*, 3 = *Started*, 4 = *Connecting*, 5 = *Run*) podle signálu *Status_del* (= signál *Status* zpožděný o polovinu periody hodinového signálu):

1. (1 => 0), (2 => 0), (3 => 0), (4 => 0) pokud *Status_del*[0] = '1'
2. (1 => 0), (2 => 0), (3 => 0), (4 => 0) pokud *Status_del*[1] = '1'
3. (1 => 0), (2 => 0), (3 => 0), (4 => 0) pokud *Status_del*[2] = '1'
4. (1 => 0), (2 => 0), (3 => 0) pokud *Status_del*[7] = '1'
5. (1 => 0), (2 => 0), (3 => 0), (4 => 0) pokud *Status_del*[8] = '1'
6. (1 => 0), (2 => 0), (3 => 0), (4 => 0) pokud *Status_del*[9] = '1'
7. (3 => 0), (4 => 0) pokud *timeout_1280ns_done* = '1'

Přechod stavů *Run => Error_Reset* bude kontrolován v následujícím testu.

3.2.4 Test detekce chyb

Tento test verifikuje detekci definovaných chyb ve stavu *Run* a následný přechod do stavu *Error_Reset*.

Průběh testu

Každá chyba bude navozena ve stavu *Run*. Signál *link_start* se nastaví do logické 1 a bude se čekat na stav *Run*. Nejprve bude navozena chyba rozpojením linky zavoláním procedury na vyčkání po dobu 1,5 mikrosekundy, kdy se nebudou měnit hodnoty vstupních signálů *Din* a *Sin* do designu. Design by měl detekovat rozpojení linky a restartovat linku. Po restartu linky bude navozena chyba parity odesláním znaku ESC se špatným paritním bitem, což by mělo způsobit znovu restart linky. Poté bude navozena úniková chyba, která proběhne celkem třikrát pro všechny nepovolené kombinace v pořadí přijatých znaků (ESC + ESC, ESC + EOP, ESC + EEP). Jako poslední bude navozena kreditová chyba, které jsou v designu definované 2. Jedna nastane, pokud si druhý konec linky vyžádá najednou více než 56 znaků, což bude dosaženo odesláním osmi FCT znaků. Druhá kreditová chyba nastane po přijetý znaků, které nebyly vyžádány designem. Tohoto bude dosaženo následovně: Do fronty pro data k odeslání ve vytvořeném SpaceWire BFM modelu bude vygenerováno celkem 57 datových znaků. Tyto znaky budou následně odeslány, ale po přijetí testovaným designem nebudou vyčítány z FIFO paměti. FIFO paměť designu má kapacitu na 64 znaků a za každých 8 míst pošle jeden znak FCT. Jelikož do FIFO paměti je při chybě v *Run* stavu vkládán znak pro chybný konec paketu a tři chyby již byly navozeny, design by si měl vyžádat pouze 56 znaků. Proto odeslání 57 znaků způsobí kreditovou chybu.

Body funkčního pokrytí

Během tohoto testu budou sledovány tyto přechody na signálu *LinkState* (0 = *Error_Reset*, 1 = *Error_Wait*, 2 = *Ready*, 3 = *Started*, 4 = *Connecting*, 5 = *Run*) podle signálu *Status_del* (= signál *Status* zpožděný o polovinu periody hodinového signálu):

1. 5 => 0 pokud *Status_del*[0] = '1'
2. 5 => 0 pokud *Status_del*[1] = '1'
3. 5 => 0 pokud *Status_del*[2] = '1'
4. 5 => 0 pokud *Status_del*[3] = '1'
5. 5 => 0 pokud *Status_del*[4] = '1'

3.2.5 Test prioritního odesílání znaků

Tento test verifikuje odesílání znaků podle priority definované ve standardu.

Průběh testu

Nejprve se nechá proběhnout inicializace linky. Po inicializaci linky bude ze SpaceWire BFM modelu odesláno 7 datových znaků. Po jejich přijetí designem budou hned vyčteny. Dále bude odeslán BFM modelem další datový znak, který bude vyčítán později. Následně bude do designu vložen datový znak k odeslání. Poté se bude čekat na logickou 1 signálu *tx_ready*. Následně bude vyčten poslední znak z FIFO paměti designu, zároveň vložen datový znak k odeslání a také časový kód k odeslání. Tímto budou připraveny k odeslání všechny možné typy znaků, protože by se stále měl odesílat první datový znak. Vyčká se na přijetí všech znaků a časového kódu. Všechny přijaté znaky budou kontrolovány, zda jsou shodné s odeslanými znaky.

Body funkčního pokrytí

Pro potřeby tohoto testu byl vytvořen 3 bitový signál *priority_check*, do jehož bitů jsou přiřazeny signály BFM modelu *got_timecode*, *got_FCT*, *got_data* takto:

priority_check[2] = *got_timecode*,
priority_check[1] = *got_FCT*,
priority_check[0] = *got_data*.

Na signálu *priority_check* bude sledována přítomnost následující sekvence:

1. (0 => 1 => 0 => 4 => 0 => 2 => 0 => 1 => 0)

3.2.6 Test komunikace

Tento test je zaměřen na dlouhodobou oboustrannou komunikaci.

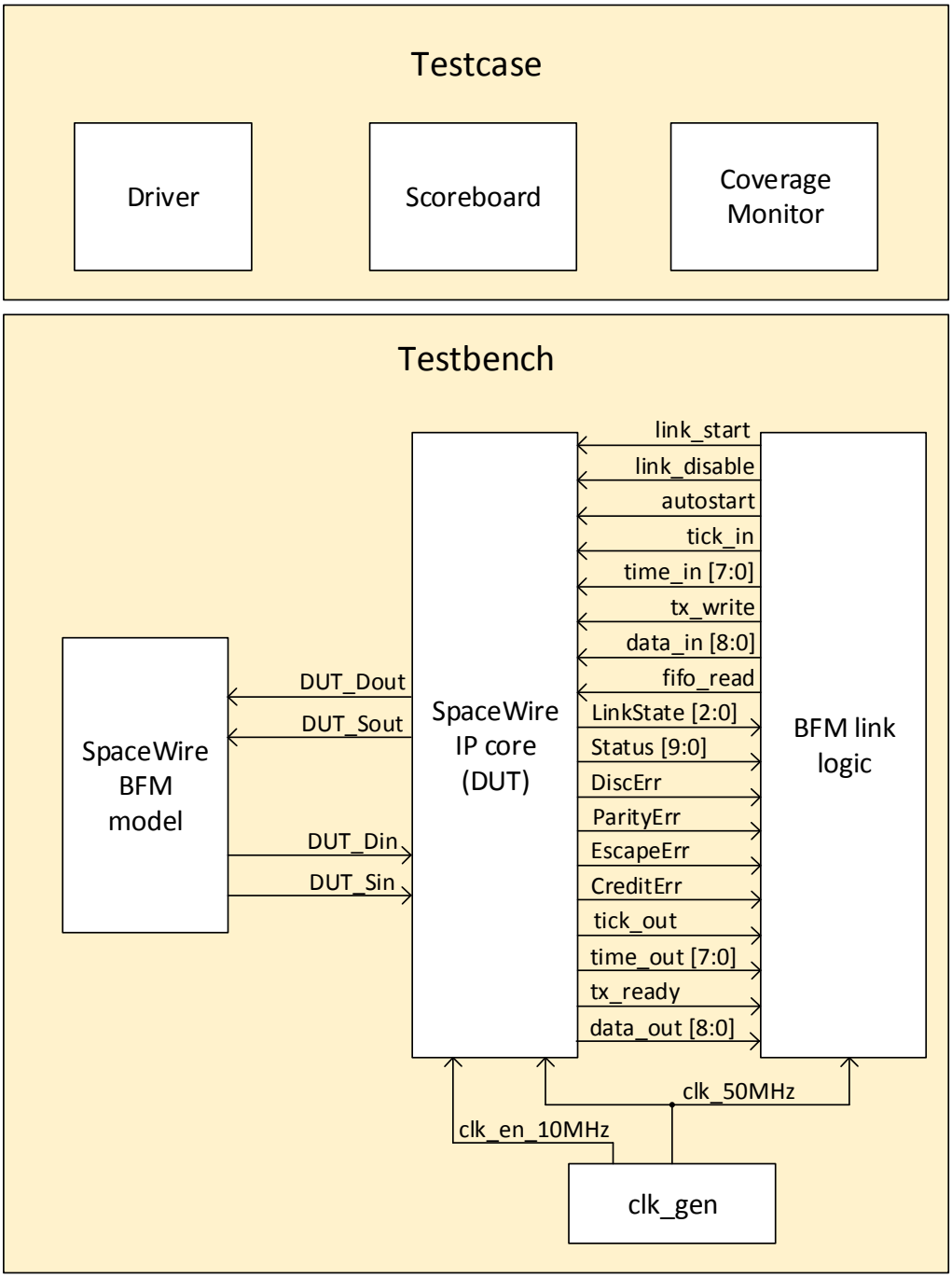
Průběh testu

Na začátku testu bude povolena inicializace linky a počká se na stav *Run* u BFM modelu i testovaného designu. Poté bude vygenerováno a odesíláno 1000 datových znaků oběma směry. Zároveň každou 1,5 µs bude vložen do BFM modelu i testovaného designu časový kód. Celkem bude vygenerováno 200 časových kódů pro odeslání oběma směry. Všechny přijaté časové kódy a datové znaky budou po přijetí vyčteny a porovnány s odeslanými znaky.

4 Popis verifikačního prostředí

Tato kapitola se věnuje popisu jednotlivých částí realizovaného verifikačního prostředí a jejich funkci.

Verifikační prostředí obsahuje 2 hlavní bloky. Prvním blokem je program Testcase, který řídí průběh verifikace. Druhý blok je modul Testbench, který vytváří prostředí pro simulaci testovaného designu. Schéma verifikačního prostředí je na obrázku 4-1.



Obrázek 4-1: Schéma verifikačního prostředí

4.1 Program Testcase

Program Testcase, jakožto vyšší vrstva verifikačního prostředí, pomocí vytvořených objektů Driver, Scoreboard a Coverage monitor řídí a kontroluje průběh simulace.

Verifikace probíhá voláním procedur v *initial* bloku programu, ve kterých je nadefinován průběh jednotlivých testů. Dále je zde volána procedura *cvg_enable,* která podle hodnoty parametru, se kterým se volána, povoluje sledování funkčního pokrytí podle probíhajícího testu, aby bylo zajištěno, že ověřovaná funkce bude splněna pouze během určeného testu. Všechny procedury vypisují do konzole právě testovanou funkci designu spolu s časem v simulaci, kdy test začíná. Na konci verifikace je do konzole vypsán celkový počet správně porovnaných znaků a počet chybných znaků.

Procedury, při kterých nejsou kontrolovány příchozí datové znaky nebo časové kódy jsou vytvořeny ve třídě Driver. Ostatní procedury jsou definovány v programu. Při generování dat je použita funkce *random* pro generaci jednoho znaku, respektive funkce *urandom_range* pro generování více znaků najednou.

Z testů popsaných ve verifikačním plánu jsou v programu definovány test prioritní odesílání znaků v proceduře *priority_sending* a test komunikace v proceduře *communication*. Pro 100% pokrytí kódu je zde nadefinováno ještě 6 dalších procedur.

První procedurou je *dut_send_data*, kde jsou do testovaného designu postupně zapisovány datové znaky k odeslání nabývající hodnot od 0 do 255 a také oba znaky pro konec paketu. Druhou přidanou procedurou je *fill_FIFO_and_read*. Zde je vygenerováno 80 datových znaků a odeslány BFM modelem do testovaného designu. Jakmile je FIFO paměť designu zaplněna, je zavolána procedura na vyčítání dat z FIFO paměti. Další procedurou je *fill_FIFO_and_EOP*, kde je vygenerováno a odesláno 63 datových znaků. Poté je odeslán znak pro konec paketu (EOP), čímž dojde k zaplnění FIFO paměti. Čtvrtou procedurou je *fill_FIFO_and_nonEOP,* kde je stejný postup jako u předchozí procedura s tím rozdílem, že posledním znakem není znak EOP. Předposlední procedurou je *fill_FIFO_and_error,* kde je znovu vygenerováno a odesláno 63 datových znaků následováno navozením chyby rozpojením linky. FIFO paměť při chybě v *Run* stavu automaticky vloží znak EEP, čímž dojde k zaplnění paměti. Přehled procedur v programu Testcase je v tabulce 4-1.

Tabulka 4-1: Procedury v programu Testcase

Procedura	Funkce
priority_sending	Test prioritního odesílání znaků
communication	Test komunikace
dut_send_data	Test odeslání všech možných hodnot datových znaků a znaků pro konec paketu
fill_FIFO_and_read	Zaplnění FIFO paměti designu a následné vyčítání znaků z paměti
fill_FIFO_and_EOP	Zaplnění FIFO paměti designu, kdy posledním znakem je znak EOP
fill_FIFO_and_nonEOP	Zaplnění FIFO paměti designu, kdy posledním znakem není znak EOP
fill_FIFO_and_error	Zaplnění FIFO paměti proběhne automatickým vložením EEP po chbě v Run stavu
bit_shift_data	Odeslání jednoho bitu následovaného kódy NULL a datovým znakem
DUT_send_data	Vložení datových znaků k odeslání testovaným designem
DUT_send_timecode	Vložení časového kódu k odeslání testovaným designem
bfm_send_timecode	Vložení časového kódu k odeslání BFM modelem

4.1.1 Řadič (Driver)

Řadič je vytvořený objekt v programu Testcase. Jsou zde nadefinovány v procedurách průběhy prvních 4 testů z verifikačního plánu a veškeré chyby, které podle stavového diagramu pro obnovu linky mohou nastat.

Průběh testu inicializace linky na příkaz signálem *link_start* je v proceduře *link_init_with_link_start*. Průběh testu inicializace linky na příkaz signálem *autostart* je v proceduře *link_init_with_autostart*. Průběh testu restartu linky podle stavového diagramu je v proceduře *link_restart*. Průběh testu detekce definovaných chyb v *Run* stavu je v proceduře *error_detection*. Přehled procedur v řadiči je v tabulce 4-2.

V procedurách s odesíláním znaků je nejprve povolena generace časové události ve vysílači BFM modelu nastavením proměnné *tx_clk_en* do logické 1. Odesílání znaků probíhá voláním procedur v top modulu BFM modelu pro odeslání jednotlivých znaků. Po dokončení těchto procedur jsou všechny signály a proměnné ve vysílači nastaveny do původní hodnoty.

Tabulka 4-2: Procedury v řadiči

Procedura	Funkce
link_init_with_link_start	Test inicializace linky na příkaz signálu link_start
link_init_with_autostart	Test inicializace linky na příkaz signálu autostart
link_restart	Test restartu linky podle stavového diagramu pro obnovu linky
error_detection	Test detekce chyb ve stavu Run
ESC_disconnect	Odeslání kombinace znaků způsobující chybu rozpojením
wrong_parity	Odeslání kombinace znaků způsobující chybu parity
send_ESC_ESC	Odeslání dvou po sobě jdoucích znaků ESC, způsobení únikové chyby
FCT_error	Odeslání znaku FCT ve stavech, kdy jeho přijetí způsobí chybu
data_char_error	Odeslání datového znaku ve stavech, kdy jeho přijetí způsobí chybu
time_char_error	Odeslání časového kódu ve stavech, kdy jeho přijetí způsobí chybu
timeout_14us	Nesplnění podmínky pro přechod do dalšího stavu do 12,8 µs
send_first_NULLs	Odeslání 2 kódu NULL

4.1.2 Výsledková tabulka (Scoreboard)

Výsledková tabulka monitoruje výstupy testovaného designu i BFM modelu a vyčtená data porovnává s očekávanými daty. Také počítá počet správných a chybných znaků.

Očekávaná data z výstupu testovaného designu jsou zapisována do fronty *SCB_FIFO_data_DUT_out* a očekávaná data z výstupu BFM modelu jsou zapisována do fronty *SCB_FIFO_data_bfm_out*. Vyčítání dat probíhá po zavolání procedury *read_DUT_output* pro data z testovaného designu a pro čtení z BFM modelu po zavolání procedury *read_bfm_output.* Procedura *read_DUT_output* po zavolání čeká na náběžnou hranu systémového hodinového signálu a pokud je signál z FIFO paměti designu v logické 0, tak zavolá proceduru *data_out_DUT_read* v bloku BFM link logic, což je nadřazený systém IP coru, pro načtení dat na výstupu a tuto hodnotu zapíše do proměnné *data_read*. Tato hodnota je pak porovnána s prvním znakem ve frontě očekávaných datových znaků z designu.

Pokud znaky souhlasí, zvýší se hodnota počítadla správných znaků o 1 a první znak ve frontě se smaže. Pokud znaky nesouhlasí, jsou do konzole vypsány očekávaná a přijatá data spolu s časem porovnávání, hodnota počítadla chyb se zvýší o 1 a první znak ve frontě je opět smazán. Jakmile ve frontě nejsou žádné znaky, procedura vyčítání znaků se ukončí. Procedura *read_bfm_output* má podobný průběh. Po jejím zavolání čeká na změnu velikosti fronty přijatých znaků *FIFO_RX* v přijímači BFM modelu. Poté následuje ve *while* cyklu vyčítání z *FIFO_RX* procedurou *FIFO_read* v top modulu BFM modelu. Procedura *FIFO_read* vyčtenou hodnotu zapíše do proměnné *data_bfm_read*, která je poté porovnána s hodnotou ve frontě očekávaných znaků z BFM modelu. Pokud znaky souhlasí, zvýší se hodnota počítadla správných znaků o 1 a první znak ve frontě se smaže. Pokud znaky nesouhlasí, jsou do konzole vypsány očekávaná a přijatá data spolu s časem porovnávání, hodnota počítadla chyb se zvýší o 1 a první znak ve frontě je opět smazán. Procedura vyčítání je opět ukončena, pokud je fronta očekávaných znaků prázdná.

Očekávané hodnoty přijatých časových kódů jsou zapsány do proměnných *expected_timecode_DUT,* respektive *expected_timecode_bfm*. Pro načtení přijatého časového kódu jsou zde procedury *read_DUT_timecode* a *read_bfm_timecode.* Procedura *read_DUT_timecode* čeká na náběžnou hranu signálu *tick_out* a zavolá proceduru pro načtení hodnoty časového kódu *timecode_DUT_out* v bloku BFM link logic, kterou uloží do proměnné *timecode_DUT*. Poté proběhne porovnání načtené a uložené hodnoty a podle výsledku se zvětší počítadlo správných nebo chybných znaků o 1. Procedura *read_bfm_timecode* čeká na náběžnou hranu signálu *got_time_code* v přijímači BFM modelu a zavolá proceduru pro načtení hodnoty časového kódu *read_timecode* v top modulu BFM modelu, kterou uloží do proměnné *timecode_bfm*. Poté proběhne porovnání načtené a uložené hodnoty a podle výsledku se zvětší počítadlo správných nebo chybných znaků o 1.

4.1.3 Monitor funkčního pokrytí (Coverage monitor)

Monitor funkčního pokrytí slouží ke sledování splnění zadaných funkcí. Je zde vytvořeno 5 skupin bodů funkčního pokrytí a procedura *cvg_enable* pro nastavení proměnné *enable,* podle jejíž hodnoty jsou povoleny jednotlivé skupiny.

Skupina *cvg_link_start_test* sleduje funkční pokrytí během testu inicializace linky na příkaz signálu *link_start.* Skupina *cvg_autostart_test* sleduje funkční pokrytí během testu inicializace linky na příkaz signálu *autostart*. Skupina *cvg_link_restart_test* sleduje funkční pokrytí během testu restartu linky. Skupina *cvg_error_detection_test* sleduje funkční pokrytí během testu detekce chyb ve stavu *Run*. Poslední skupina *cvg_priority_sending_test* sleduje funkční pokrytí během testu prioritního odesílání znaků.

4.2 Modul Testbench

Modul Testbench je top modul nižší vrstvy verifikační prostředí a vytváří prostředí pro simulaci testovaného designu. Obsahuje vytvořený Spacewire BFM model, nadřazený systém pro testovaný design BFM link logic, testovaný SpaceWire IP core a generátor hodinových signálů. Také je zde vytvořen reset pro testovaný design a BFM model.

4.2.1 Generátor hodinových signálů (clk_gen)

Generátor hodinových signálů je jednoduchý blok obsahující jednu proceduru a dva *always* bloky.

Procedura *clk_run*, volaná z programu Testcase, povoluje generaci hodinových signálů. Pokud je volána s hodnotou 1, je generování povoleno. Volání s hodnotou 0 ukončí generaci hodinového signálu.

První *always* blok generuje povolovací signál o frekvenci 10 MHz. Délka povolovacího pulzu je jedna perioda hlavního hodinového signálu. Ten je pro simulaci nastaven na frekvenci 50 MHz a je generován v druhém *always* bloku.

4.2.2 Nadřazený modul pro IP core (BFM link logic)

Nadřazený modul slouží k ovládání linky, zapisování a vyčítání dat a časových kódů z IP coru. Obsahuje jeden *always* blok a 5 procedur.

V *always* bloku je zpožďován signál *status* o jednu polovinu periody hodinového signálu pro potřeby funkční verifikace. Zpožděný signál je pojmenován *status_del*.

V proceduře *link_control* jsou nastavovány ovládací signály linky *link_start, link_disable* a *autostart*.

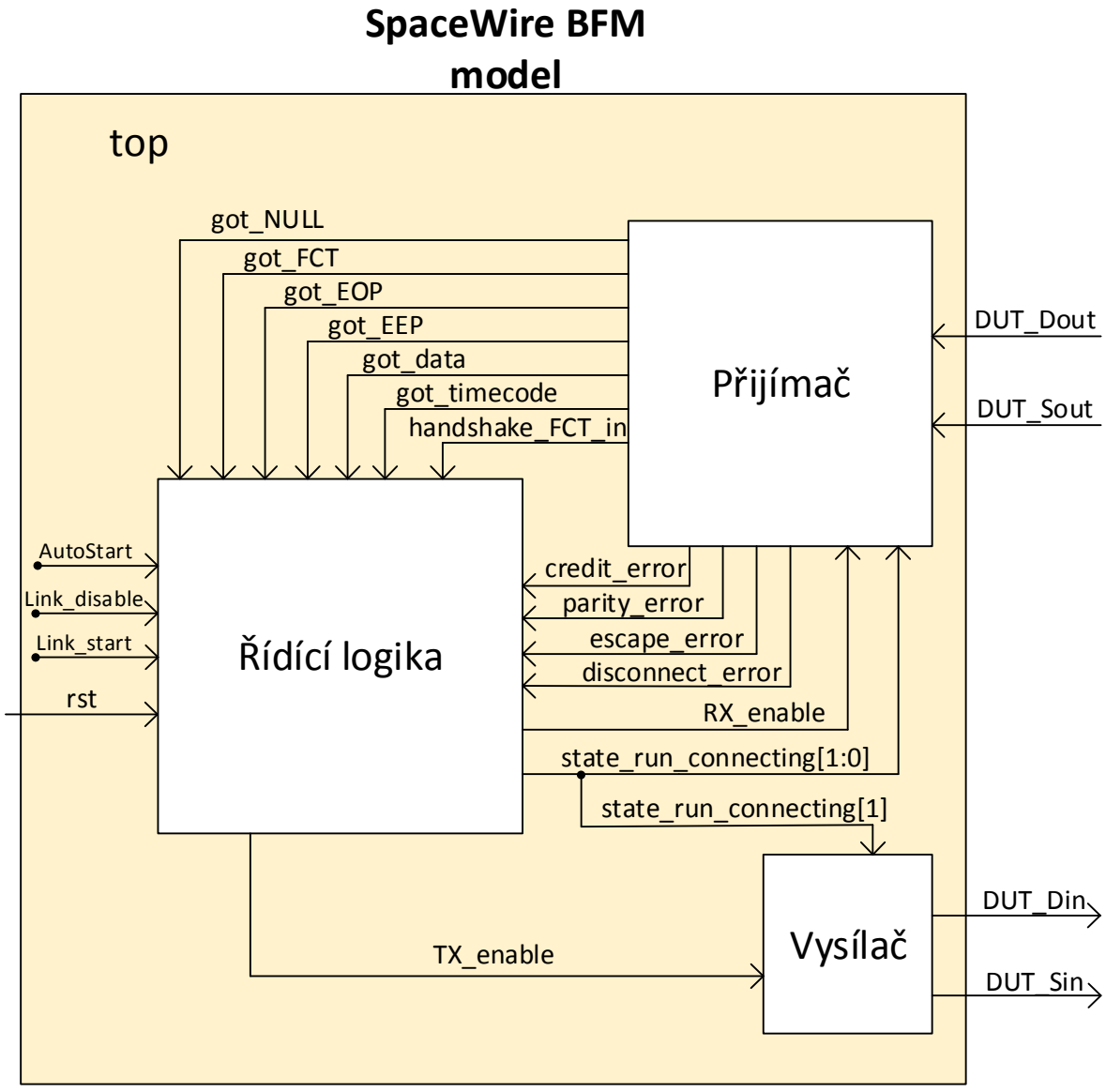
Další 2 procedury se starají o zápis a načtení časového kódu. Zápis časové kódu probíhá v proceduře *timecode_to_DUT*. Po zavolání procedury je nastaven signál *tick_in* do logické 1 a hodnota časového kódu na vstupu *time_in*. Poté se čeká na náběžnou hranu a poté sestupnou hranu hodinového signálu, kdy je nastaven signál *tick_in* do logické 0. Načtení časového kódu probíhá v proceduře *timecode_DUT_out*, která vrací načtenou hodnotu.

Poslední 2 procedury slouží k zápisu a vyčtení dat. Zde je rozdíl u testovaného designu a vytvořeného BFM modelu. U testovaného designu je kontrolní vlajka datového znaku v nejméně významném bitu, kdežto u BFM modelu je kontrolní vlajka nejvýznamnější bit. Procedura *data_to_DUT_write* zapisuje data do designu k odeslání. Nejprve nastaví signál *tx_write* do logické 1 a poté zkontroluje zapisovaná data. Pokud má odeslat znak pro konec paketu nebo znak pro chybný konec paketu, nastaví na vstupu *data_in* nejnižší bit do logické 1 a ostatní do logické 0 respektive dva nejnižší bity do logické 1 a ostatní do logické 0. Pokud je zapisován datový znak, nastaví na vstupu *data_in* nejnižší bit do logické 0 a ostatní podle

hodnoty zapisovaného znaku. Data jsou vyčtena zavoláním procedury *data_out_DUT_read*. Signál *fifo_read* je nastaven do logické 1 a na sestupné hraně hodinového signálu je kontrolován typ znaku na výstupu. Pokud je to znak pro konec paketu, vrací procedura hodnotu 256 = "100000000". U znaku pro chybný konec paketu vrací procedura hodnotu 257 = "100000001". Pro datový znak vrací procedura hodnotu ({1'b0,data_out[8:1] }), kde nejnižší bit je posunut na pozici nejvyššího bitu a ostatní bity jsou posunuty o 1 pozici níže.

4.2.3 SpaceWire BFM model

SpaceWire BFM model je vytvořen jako bus functional model (neboli sběrnicový funkční model), což znamená, že pokud se zaměříme pouze na výstupy modelu, tak se chová stejně jako předloha. Pro snadnější přístup jsou procedury volané z vyšší vrstvy verifikačního prostředí také v top modulu BFM modelu. Schéma SpaceWire modelu je na obrázku 4-2.



Obrázek 4-2: Schéma SpaceWire BFM modelu

Řídící logika (bfm_fsm)

Řídící logika je jednoduchý stavový automat skládající se z 3 *always* bloků. První *always* blok je časovač přechodů mezi stavy stavového automatu. Reaguje na změny signálu *current_state* (stav stavového automatu), *timer_en* (povolení časovače) a *rst* (reset signál). Pomocí konstrukce *fork - join_any* je rozdělen do 3 procesů:

1) Pokud je *timer_en* v logické 1 a reset není aktivní, tak podle aktuálního stavu *current_state* probíhá odpočet. Ve stavu *Error_Reset* se odpočítává 6,4 μs. Ve stavech *Error_Wait, Started* a *Connecting* se odpočítává 12,8 μs. Po doběhnutí odpočtu se nastaví signál *timer_done* do logické 1 na dobu 10 ns.

2) Druhý proces čeká na sestupnou hranu *timer_en*. Pokud nastane, je první proces ukončen.

3) Třetí proces čeká na nástupnou hranu reset signálu. Stejně jako v předchozím procesu, pokud nastane náběžná hrana, je časovač ukončen.

Druhý *always* blok slouží jako kombinační logika přechodu stavů stavového automatu. V jeho citlivostním seznamu jsou všechny vstupní signály do řídící logiky z obrázku 4-2 a také signál *timer_done* z prvního *always* bloku. Podmínky přechodů mezi stavy jsou stejné jako na Obrázek 1-5.

Třetí *always* blok nastavuje výstupy z řídící logiky. Po přechodu do stavu *Connecting* je zavolána procedura *counters* ve vysílači na povolení odeslání jednoho FCT znaku pro inicializaci linky. Pokud ve stavu *Run* indikuje přijímač přijetí FCT znaku, je zavolána stejná procedura pro navýšení počtu N-znaků (data nebo konce paketu) vyžádaných druhou stranou linky. Hodnoty jednotlivých výstupů podle stavu řídící logiky jsou v tabulce 4-3.

Tabulka 4-3: Hodnoty výstupních signálů z řídící logiky

Výstup	**Stav**					
	Error_Reset	***Error_Wait***	***Ready***	***Started***	***Connecting***	***Run***
RX_enable	'0'	'1'	'1'	'1'	'1'	'1'
TX_enable	'0'	'0'	'0'	'1'	'1'	'1'
handshake_FCT_out	'0'	'0'	'0'	'0'	'1'	'1'
state_run_connecting	"00"	"00"	"00"	"00"	"01"	"10"

Vysílač (bfm_tx)

Vysílač se stará o zakódování znaků k odeslání a jejich následné odesílání. Obsahuje 3 *always* bloky a 15 procedur.

První *always* blok se stará o generaci časové události, pokud je *TX_enable* nebo *tx_clk_en* v logické 1. Aktivní je vždy maximálně jeden povolovací signál. Při normální operaci vysílače je *TX_enable* v logické 1 a probíhá generace hodinové události nejprve s frekvencí 10 MHz. Po přechodu řídící logiky do *Run* stavu, indikující signálem *state_run,* se změní rychlost hodinové události na 50 MHz, jakmile není posílán žádný znak. O toto přepínání rychlosti se stará druhý *always* blok.

Poslední *always* blok je hlavním blokem, který volá procedury odesílání znaků podle priority ve standardu. Je rozdělen na dva procesy:

A) První proces probíhá neustále, pokud je *TX_enable* v logické 1 a volá procedury podle následující priority:

   1) Pokud je k dispozici časový kód (*time_code_en* = '1') zavolá proceduru k odeslání časového kódu *send_time_code* s hodnotou v proměnné *time_code*.

   2) Pokud není k dispozici časový kód a počítadlo *FCT_counter* má větší hodnotu než 0, tak zavolá proceduru *send_FCT* k odeslání znaku FCT a hodnotu počítadla *FCT_counter* zmenší o 1.

   3) Pokud není k dispozici časový kód ani FCT znak, tak se kontroluje, zda si druhá strana linky vyžádala znaky, *N_char_counter* je větší jak '0', a ve fronta znaků k odeslání není prázdná. Podle první hodnoty ve frontě jsou odeslány tyto znaky: (0-255) - datový znak, 256 – znak EOP, 257 – znak EEP, 258 – znak ESC s nesprávným paritním bitem, 259 – dva znaky ESC, 260 – znak ESC následovaný znakem EOP, 261 – znak ESC následovaný znakem EEP, 262 – přerušení komunikace na dobu 1,3 μs.

   4) Pokud není žádný znak k dispozici, je odesílán kontrolní kód NULL pro udržení aktivní linky.

B) Druhý proces čeká na sestupnou hranu *TX_enable,* což funguje jako reset.

Navyšování hodnot počítadel pro prioritní odesílání probíhá v proceduře *counters,* podle volaného parametru: *"time_code"* změní hodnotu *time_code_en* na '1', *"FCT"* navýší hodnotu *FCT_counter* o 1, *"N_char"* navýší hodnotu *N_char_counter* o 8. Dále je zde procedura na zápis hodnot do fronty znaků k odeslání *FIFO_write*, která vloží hodnotu na konec fronty, a procedura vložení hodnoty časového kódu k odeslání *time_code_write* do proměnné *time_code*.

Ostatní procedury slouží k odesílání znaků. Mají 2 vstupně-výstupní proměnné *last_data_bit* (poslední bit předchozího znaku) a *parity* (paritní bit). Pokud je odesílán znak s kontrolní vlajkou v '1', tak se změní hodnota paritního bitu, aby byla dodržena lichá parita. Dále jsou kontrolovány znaky v tomto pořadí {*last_data_bit*, *parity*, bity odesílaného znaku}, pokud mají dva po sobě jdoucí bity jinou hodnotu, tak se změní hodnota *D_int* (výstupní datový signál). Pokud mají stejnou hodnotu, změní se hodnota *S_int* (výstupní strobe signál). Po dokončení odesílání znaku vrátí procedura hodnotu posledního datového bitu a hodnotu paritního bitu pro další odesílaný znak. V procedurách odesílání znaků se nastavuje ještě dvoubitový signál *sending_char*, jehož nižší bit značí odesílání jednoduchého znaku a vyšší bit značí odesílání složeného znaku (odesílání se značí logickou 1). Přehled procedur ve vysílači je v tabulce 4-4.

Tabulka 4-4: Procedury ve vysílači

Procedura	Funkce
counters	Nastavení počítadel FCT a datových znaků a časových kódů k odeslání
FIFO_write	Zápis dat do fronty znaků k odeslání
time_code_write	Zápis hodnoty časového kódu k odeslání
send_time_code	Odeslání časového kódu
send_NULL	Odeslání kontrolního kódu NULL
send_ESC_EOP	Odeslání znaku ESC následovaného znakem EOP
send_ESC_EEP	Odeslání znaku ESC následovaného znakem EEP
send_double_ESC	Odeslání dvou znaků ESC
disconnect	Přerušení komunikace po dobu 1,3 µs
send_data_char	Odeslání datového znaku
send_ESC_wrong_parity	Odeslání ESC znaku s nesprávným paritním bitem
send_FCT	Odeslání FCT znaku
send_EOP	Odeslání EOP znaku
send_EEP	Odeslání EEP znaku
send_ESC	Odeslání ESC znaku

Přijímač (bfm_rx)

Přijímač se stará o přijímání a dekódování příchozích znaků a detekci všech chyb na lince. Přijetí znaků nebo detekci chyby indikuje řídící logice pulsem na příslušném signálu. Hodinový signál přijímače je rekonstruován z příchozích signálů *Din (DUT_Dout)* a *Sin (DUT_Sout)* použitím fuknce xor.

V jednom *always* bloku probíhá kontrola kreditní chyby a vyžádání datových znaků zavoláním procedury *counters("FCT")* k navýšení počítadla FCT znaků k odeslání ve vysílači za každých 8 volných míst ve frontě přijatých znaků. Další *always* blok hlídá rozpojení linky. Blok je rozdělen na dva procesy: jeden sleduje rekonstruovaný hodinový signál *rx_clock* a druhý odpočítává dobu 850 ns. Pokud jsou přijata data, nastane změna na hodinovém signálu *rx_clock* a odpočet se resetuje.

Ve třetím *always* bloku probíhá hlavní činnost přijímače. Blok je opět rozdělen na dva procesy. První proces probíhá, pokud je *RX_enable* v logické 1. Nejprve čeká na nástupnou hranu vstupního signálu *Sin*, protože úvodní sekvence by měla takto začínat, a je povoleno vzorkování vstupních signálů a detekce chyby rozpojení linky. Jsou načteny první dva bity, kterými jsou paritní bit a kontrolní vlajka, podle které se určí počet dalších načítaných bitů. Po načtení prvních dvou bitů je signál *get_new_char* nastaven do logické 0 a je zavolána procedura na kontrolu parity *parity_check*. Podle hodnoty signálu *char_check_select* se kontroluje parita kontrolního znaku ('0') nebo datového znaku ('1'). Pokud je parita v pořádku, je zavolána procedura *FIFO_push_NULL_decode_Escape_error*, kde se kontroluje podle kombinace přijatých znaků, zda se jedná o časový kód nebo datový znak, byl přijat kód NULL nebo nastala úniková chyba nebo jde jen o samostatný kontrolní znak. Po proběhnutí této procedury se podle hodnoty kontrolní vlajky určí, zda další bity budou patřit ke kontrolnímu znaku nebo půjde o datový znak. V případě kontrolní vlajky v '0' následují dva bity kontrolního znaku a je zavolána procedura *control_char_decode,* kde se určí, o jaký kontrolní znak se jedná, *char_check_select* je nastaven na hodnotu '0' a *get_new_char* je nastaven na hodnotu '1', protože další bity budou patřit k dalšímu znaku. Pokud je kontrolní vlajka v '1' následuje 8 bitů datového znaku, *char_check_select* je nastaven na hodnotu '1' a *get_new_char* je nastaven na hodnotu '1', protože další bity budou patřit k dalšímu znaku. Poté se celý proces opakuje, dokud je *RX_enable* v logické 1. Procedura kontroly parity následuje hned po načtení paritního bitu a kontrolní vlajky, protože paritní bit a kontrolní vlajka patří k paritě předchozího znaku. Druhý proces čeká na sestupnou hranu signálu *RX_enable*, což funguje jako reset.

V tabulce 4-5 je přehled procedur v přijímači.

Tabulka 4-5: Procedury v přijímači

Procedura	Funkce
FIFO_read	Vyčtení z fronty přijatých znaků
control_char_decode	Dekódování přijatého kontrolního znaku
parity_check	Kontrola parity
FIFO_push_NULL_decode_Escape_error	Dekódování složených znaků podle kombinace přijatých znaků, kontrola únikové chyby

5 Závěr

Úkolem této bakalářské práce bylo verifikovat SpaceWire IP core vytvořeného na Ústavu mikroelektroniky. Za tímto účelem bylo v jazyce SystemVerilog vytvořeno vrstvové verifikační prostředí. V nejnižší vrstvě byl kromě testovaného designu vytvořen model koncového bodu sítě a také nadřazený systém pro komunikaci respektive ovládání testovaného designu.

Dále bylo vytvořeno 5 hlavních testů k ověření funkčnosti designu. Těmito testy se ověřovala schopnost inicializovat linku na příkaz signálu *link_start* a *autostart*, ověření všech možných přechodů stavového automatu v řídící logice designu, detekce všech definovaných chyb ve stavu *Run*, odesílání znaků podle definované priority a schopnost dlouhodobé obousměrné komunikace s kontrolou přijatých a odeslaných znaků. V těchto testech dosáhl testovaný design 100% funkčního pokrytí. Pokrytí kódu však kompletní nebylo. Proto bylo dopsáno 6 dalších testů, po kterých již bylo 100% pokrytí kódu.

Použité zdroje

[1] Parkes, S. M. *SpaceWire User´s Guide,* STAR-Dundee Limited, 2012, ISBN 978-0-9573408-0-0

[2] SpaceWire CODEC [online]. European Space Agency [cit. 2015-12-03]. Dostupné z WWW: <http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SpaceWire-b_-_HDL>

[3] European Cooperation for Space Standardization, Standard ECSS-E-ST-50-12C, *SpaceWire, Links, Nodes, Routers and Networks,* 2nd issue, European Cooperation for Space Data Standardization, July 2008

[4] Texas Instruments, *LVDS Owner´s Manual* [online]. 2008 [cit. 2015-11-07]. Dostupné z WWW: <http://www.ti.com/lit/ml/snla187/snla187.pdf>.

[5] SystemVerilog Verification, *Testbench* [online], [cit. 2015-12-02]. Dostupné z WWW: <http://www.testbench.in/TS_00_INDEX.html>.

[6] What is SystemVerilog?, *Doulos* [online], [cit. 2015-12-02]. Dostupné z WWW: <https://www.doulos.com/knowhow/sysverilog/whatissv/>.

Seznam příloh

A Výsledná zpráva funkčního pokrytí a pokrytí kódu po verifikaci

```
Coverage Report Summary Data by file

================================================================================
=== File: /home/users/stud50/projects/DUT/clock_recovery.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            1         1         0     100.0
    FEC Expression Terms             2         2         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/control_logic.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           74        74         0     100.0
    Branches                        33        33         0     100.0
    FEC Condition Terms             26        26         0     100.0
    FEC Expression Terms             9         9         0     100.0
    FSMs                                                     100.0
        States                       6         6         0     100.0
        Transitions                 16        16         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/disconnect_check.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           30        30         0     100.0
    Branches                        21        21         0     100.0
    FEC Condition Terms              4         4         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/init_timer.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           15        15         0     100.0
    Branches                        13        13         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/receiver.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            1         1         0     100.0
    FEC Expression Terms             2         2         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/rx_buffer_fifo.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           72        72         0     100.0
    Branches                        42        42         0     100.0
    FEC Condition Terms             14        14         0     100.0
```

```
================================================================================
=== File: /home/users/stud50/projects/DUT/rx_char.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           57        57         0     100.0
    Branches                        27        27         0     100.0
    FEC Condition Terms              2         2         0     100.0
    FEC Expression Terms            10        10         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/rx_front.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           53        53         0     100.0
    Branches                        22        22         0     100.0
    FEC Expression Terms             2         2         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/transmitter.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            1         1         0     100.0
    Branches                         2         2         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/tx_front.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           51        51         0     100.0
    Branches                        28        28         0     100.0
    FEC Condition Terms              2         2         0     100.0

================================================================================
=== File: /home/users/stud50/projects/DUT/tx_fsm.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                          101       101         0     100.0
    Branches                        47        47         0     100.0
    FEC Condition Terms              4         4         0     100.0
    FEC Expression Terms            16        16         0     100.0
    FSMs                                                       100.0
        States                       3         3         0     100.0
        Transitions                  7         7         0     100.0


TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 5

Total Coverage By File (code coverage only, filtered view): 100.0%
```