VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VERIFIKACE SYSTÉMU PRO DETEKCI NEŽÁDOUCÍHO PROVOZU

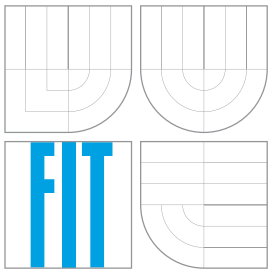
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VLASTIMIL KOŠAŘ

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VERIFIKACE SYSTÉMU PRO DETEKCI NEŽÁDOUCÍHO PROVOZU

VERIFICATION OF INTRUSION DETECTION SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VLASTIMIL KOŠAŘ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ TOBOLA
SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Košař Vlastimil**
Obor: Informační technologie
Téma: **Verifikace systému pro detekci nežádoucího provozu**
Kategorie: Návrh číslicových systémů

Pokyny:

1. Seznamte se s architekturou systému pro detekci nežádoucího provozu v FPGA.
2. Seznamte se s protokolem IPv6. Navrhněte a implementujte rozšíření systému pro detekci nežádoucího provozu o podporu tohoto protokolu.
3. Prostudujte možnosti pokročilého testování a verifikace pomocí jazyka System Verilog.
4. Navrhněte obecné prostředí pro verifikaci architektury systému v FPGA.
5. Implementujte navrženého systém a proveďte verifikaci systému pro detekci nežádoucího provozu.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího rozšíření systému.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních dvou bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Tobola Jiří, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 00 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Licenční smlouva je uvedena v archivním výtisku uloženém v knihovně FIT VUT v Brne.

Abstrakt

Tato práce pojednává o verifikaci systému pro detekci nežádoucího provozu a jeho rozšíření o podporu protokolu IPv6. Jsou zde popsány možnosti jazyka System Verilog pro verifikaci, vybranná metodologie verifikace, výhody a nevýhody různých přístupů k verifikaci a testování. Je navržena struktura systému pro verifikaci klíčových částí systému pro detekci nežádoucího provozu, jehož klíčovou součástí je paketový generátor.

Klíčová slova

Verifikace, SystemVerilog, TCP/IP, Klasikátor, Paketový generátor, IDS

Abstract

This thesis focuses on verification of Intrusion Detection System and its IPv6 support extension. Here are described posibilities of SystemVerilog for verification, choosen verification methodology, pros and cons of different verification and testing approaches. Here is designed structure of verification of key parts of Intrusion Detection System. The key component of verification system is Packet Generator.

Keywords

Verification, SystemVerilog, TCP/IP, Classifier, Packet generator, IDS

Citace

Vlastimil Košař: Verifikace systému pro detekci nežádoucího provozu, bakalářská práce, Brno, FIT VUT v Brně, 2008

Verifikace systému pro detekci nežádoucího provozu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Toboly. Odbornou pomoc mi dále poskytl Petr Kobierský. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vlastimil Košař
14. května 2008

Poděkování

Chtěl bych poděkovat všem, kteří mi poskytli odbornou pomoc.

Obsah

1 Úvod **3**

2 Systém pro detekci nežádoucího provozu **4**
2.1 Úvod . . . 4
2.2 HFE_C . . . 5
2.2.1 Popis . . . 5
2.2.2 Rozhraní . . . 5
2.3 Ptrn-match . . . 5
2.3.1 Popis . . . 5
2.3.2 Rozhraní . . . 7
2.4 Klasifikátor . . . 7
2.4.1 Popis . . . 7
2.4.2 Rozhraní . . . 7

3 Rozšíření o podporu IPv6 **8**
3.1 Úvod . . . 8
3.2 IPv6 z pohledu klasifikátoru . . . 8
3.3 Adresování v IPv6 . . . 8
3.4 Návrh a implementace . . . 9

4 Verifikační metodologie **10**
4.1 Úvod . . . 10
4.2 Místa výskytu chyb . . . 10
4.3 Metody Verifikace . . . 10
4.3.1 Formální verifikace . . . 10
4.3.2 Řízené testování . . . 11
4.3.3 Constrined-random test . . . 11

5 SystemVerilog **17**
5.1 Úvod . . . 17
5.2 Datové typy . . . 17
5.2.1 Základní datové typy . . . 17
5.2.2 Pole . . . 17
5.2.3 Fronta . . . 18
5.2.4 Asociativní pole . . . 18
5.2.5 Metody polí . . . 18
5.2.6 Uživatelsky definované typy . . . 18
5.2.7 Řetězce . . . 19

5.3 Objektově orientované programování . . . 19
5.4 Randomizace . . . 19
5.4.1 Úvod . . . 19
5.4.2 Prostředky SystemVerilogu pro randomizaci . . . 20
5.5 Vlákna a meziprocesová komunikace . . . 21
5.5.1 Vlákna . . . 21
5.5.2 Meziprocesová komunikace . . . 21
5.6 Function coverage . . . 22
5.6.1 Code coverage . . . 22
5.6.2 Assertion coverage . . . 22
5.6.3 Zásady pro použití function coverage . . . 22

6 Verifikace systému pro detekci nežádoucího provozu 23
6.1 Návrh . . . 23
6.1.1 Systém . . . 23
6.1.2 Paketový generátor . . . 24
6.1.3 Podporované síťové protokoly . . . 24
6.1.4 Infrastruktura pro nestandardní rozhraní . . . 25
6.1.5 Scoreboardy . . . 26
6.2 Implementace . . . 26
6.2.1 PacketGenerator . . . 26
6.2.2 Třída Layer . . . 28
6.2.3 Ethernet_II . . . 30
6.2.4 Ethernet_II_dot1q . . . 30
6.2.5 IPv4 . . . 31
6.2.6 IPv6 . . . 32
6.2.7 ICMP . . . 32
6.2.8 ICMPv6 . . . 33
6.2.9 TCP . . . 33
6.2.10 UDP . . . 34
6.2.11 RAW . . . 35
6.2.12 RAWPattern . . . 35
6.2.13 Další třídy . . . 35
6.2.14 Stav implementace systému . . . 35

7 Závěr 36
7.1 Možná rozšíření . . . 36
7.1.1 Podpora dalších protokolů . . . 36
7.1.2 Automatické generování modelu jednotky HFE_C . . . 36
7.1.3 Automatické zpracování pravidel systému Snort . . . 36
7.2 Zhodnocení . . . 36

A Detailní schéma systému pro detekci nežádoucího provozu 42

B Diagram tříd síťových protokolů 44

C Seznam použitých zkratek 46

D Návod na spuštění 47

Kapitola 1

Úvod

Tato práce se zabývá rozšířením systému pro detekci nežádoucího provozu o podporu protokolu IPv6 a následnou verifikaci systému. V průběhu vytváření nové verze systému pro detekci nežádoucího provozu nastala potřeba vytvořit efektivnější způsob ověřování správné funkce systému, než byl používán doposud. Jako nejvhodnější se ukázal být jazyk System Verilog, jež je používán pro verifikaci v komerční sféře. Jedná se o objektově orientovaný jazyk, jenž obsahuje mnoho součástí užitečných při verifikaci.

Formální verifikace se v současném době ještě není dostatečně vhodná pro účely průběžné verifikace VHDL kódu. Také vyžaduje rozsáhlé znalosti zejména temporální logiky, která je vyžadována pro vytvoření formálního modelu pro v současnosti jedinou automatizovatelnou metodu formální verifikace - Model checking.[4] Proto se v této práci zaměřuji na metodologii Constrained-random test, které sice není formální, ale přesto dosahuje dobrých výsledků.

Práce je členěna do pěti kapitol. V kapitole Systém pro detekci nežádoucího provozu je popsán systém pro detekci nežádoucího provozu, zejména s přihlédnutím ke změnám v aktuální verzi oproti jeho popisu v [16]. Následující kapitola pojednává o rozšíření systému pro detekci nežádoucího provozu o podporu protokolu IPv6. Možnosti jazyka System Verilog jsou probrány v kapitole následující. Poté je popsána vybraná metodologie verifikace a popsán návrh verifikace a jeho realizace. V závěru jsou diskutovány dosažené výsledky a navržena budoucí možná rozšíření systému.

Tato práce vznikla v rámci projektu Liberouter, součásti výzkumného záměru sdružení CESNET z. s. p. o.

Kapitola 2

Systém pro detekci nežádoucího provozu

2.1 Úvod

Systém pro detekci nežádoucího provozu je určen pro rozpoznání probíhajícího útoku a výskytu nežádoucího provozu na síti, jako jsou například viry, trojští koně, apod. IDS je určen pro akceleraci práce open-source systému pro detekci nežádoucího provozu Snort [18]. Tento systém používá pro detekci nežádoucího provozu pravidla, která obsahují mnoho údajů popisujících útok či nežádoucí provoz. Jsou to například síťové protokoly, IP adresy, porty, řetězce vyskytující se v paketech. Tato pravidla pokrývají většinu známých útoků a virů. Při zpracování paketů zabere více než 80% času vyhledávání řetězců.[17] Z tohoto důvodu vznikl systém IDS, který je určen k hardwarové akceleraci vyhledávání řetězců v paketech. Toto řešení umožňuje zpracovávat pakety na gigabitových sítích tím způsobem, že do aplikace Snort pošle jen podezřelé pakety k jejich další analýze.

Systém IDS je založen na hardwarové kartě COMBO6X s interfacovou kartou SFPRO. Tyto karty obsahují mimo jiné FPGA čipy Virtex-II Pro. Pravidla systému Snort jsou přeložena do VHDL kódu, který zajišťuje vyhledání řetězců v paketech. Pakety jsou také klasifikovány podle nejdůležitějších příznaků jako jsou zdrojová a cílová IP adresa, protokol transportní vrstvy a zdrojové a cílové porty. Je použit synchronizační kmitočet 100 Mhz, přičemž maximální propustnost dosahuje až 3.2 Gb/s. [17]

Design systému je založen na vrstevnaté struktuře složené z části hardwarové a části softwarové. Hardwarová vrstva se skládá z vrstvy fyzické (COMBO6x a SFPRO karta) a z vrstvy firmwarové (obsah FPGA čipů). Softwarová část se skládá z ovladačů, díky kterým se chová jako obyčejná síťová karta, z knihoven a nástrojů pro nahrávání designu, jeho ovládání a ladění. Dále se budeme zabývat pouze vrstvou firmwaru.

Původní systém byl založen na command protokolu, který tvořil sběrnici mezi jednotkami designu. Jako sběrnice systémová byla použita stará lokální sběrnice. Více informací o této verzi systému je dostupných například v [16] [17]. S příchodem platformy NetCOPE (viz[32]) nastala potřeba tento systám upravit pro tuto platformu. Jelikož je na této platformě používán flexibilnější systém sběrnic skládající se ze sběrnice FrameLink [30] a ze systému propojovacích sběrnic, musely být komponenty přepsány na tyto sběrnice. Pro FrameLink existuje velké množství hotových komponent, což urychluje vývoj, na rozdíl od command protokolu, na kterém existovaly různé příkazy [16], tudíž jednotlivé komponenty spolu nemusely být kompatibilní.

Systém se skládá z následujících důležitých jednotek. IBUF, jenž je součástí platformy Netcope, který slouží pro příjem paketů ze čtyř síťových interface, přičemž je kontrolována jejich délka a CRC. Pakety, které projdou jsou poslány do jednotky HFE_C, která slouží k extrahování vybranných hlaviček. Tyto hlavičky jsou zaslány do klasifikační jednotky, která porovná extrahované hlavičky s uloženými pravidly systému Snort. Tato jednotka slouží k omezení počtu nalezených paketů s nějakým řetězcem, poněvadž se může vyskytovat i v jiných paketech než odpovídají danému pravidlu. Paket je zaslán jednotce pro vyhledávání řetězců. Pokud je paket rozpoznán jako nežádoucí, pak je zaslán k další analýze systému Snort. Kompletní schéma systému je možné najít v přílohách na obr. A.

2.2 HFE_C

2.2.1 Popis

Komponenta HFE_C slouží k extrakci položek z hlaviček síťových protokolů. Je založena na konečném automatu, tudíž je schopna zpracovat v každém taktu jedno slovo paketu. Je napsána v jazyce Handel-C jenž je modifikací jazyka C pro popis paralelních struktur. Tato jednotka je modifikovatelná, v době překladu je možné určit, které položky se budou extrahovat [31]. Pro IDS je tato jednotka nastavena tak, aby zpracovávala pakety s IPv4 a IPv6 protokoly síťové vrstvy (včetně ICMP) a s protokoly transportní vrstvy TCP a UDP. Paket, který neodpovídá tomuto nastevení je extrahován jako neznámý protokol síťové a nebo transportní vrstvy.

2.2.2 Rozhraní

Vstupem jednotky je 16 bitová sběrnice FrameLink. Rámec přenášený po této sběrnici se skládá ze dvou částí. V první části, která má velikost 32 bitů obsahuje identifikaci interface, který přijal paket a velikost paketu v bytech. V následující části se vyskytuje samotný paket.

Jednotka má dvě výstupní rozhraní. Výstupem rozhraní STORAGE je 16 bitová sběrnice FrameLink obsahující paket.Přenášený rámec má 3 části. V první části, která má velikost 32 bitů obsahuje identifikaci interface, který přijal paket a velikost paketu v bytech. Ve druhé části jsou uloženy hlavičky síťových protokolů. V poslední části jsou uložena užitečná data paketu, ve kterém jsou následovně hledány řetězce. Výstupem rozhranní UH je 32 bitová sběrnice FrameLink obsahující extrahované položky z hlaviček ve formátu unifikovaných hlaviček. Unifikovaná hlavička se skládá z 16 bitové adresy hlavičky a 16 bitů extrahovaných dat. Seznam všech podporovaných unifikovaných hlaviček s jejich popisem je možné nalézt v tabulce 2.1.

2.3 Ptrn-match

2.3.1 Popis

Tato jednotka slouží k vyhledávání vzorů v datech paketu. Vzory nejsou hledány v hlavičkách síťových protokolů. Kód jednotky je generován na základě pravidel systému Snort. Řetězec je převeden na rozšířený nedeterministický konečný automat, který zpracovává čtyři znaky v jednom taktu. Tento automat je pak převeden do ekvivalentní VHDL podoby, která se skládá z klopných obvodů a prvků AND a OR. Tímto je možné dosáhnou propustnosti

ID	Adresa hlavičky	popis
1	0Eh	identifikace interface (první hlavička která přijde)
2	00h	verze IP
3	01h	Zdrojová IP adresa (IPv4 i IPv6)
4	02h	Zdrojová IP adresa (IPv4 i IPv6)
5	03h	Zdrojová IP adresa (IPv6)
6	04h	Zdrojová IP adresa (IPv6)
7	05h	Zdrojová IP adresa (IPv6)
8	06h	Zdrojová IP adresa (IPv6)
9	07h	Zdrojová IP adresa (IPv6)
10	08h	Zdrojová IP adresa (IPv6)
11	09h	Cílová IP adresa (IPv4 i IPv6)
12	0Ah	Cílová IP adresa (IPv4 i IPv6)
13	0Bh	Cílová IP adresa (IPv6)
14	0Ch	Cílová IP adresa (IPv6)
15	0Dh	Cílová IP adresa (IPv6)
16	0Eh	Cílová IP adresa (IPv6)
17	0Fh	Cílová IP adresa (IPv6)
18	10h	Cílová IP adresa (IPv6)
19	11h	Protokol transportní vrstvy (viz 2.2)
20	12h	Zdrojový port (TCP, UDP)
21	13h	Cílový port (TCP, UDP)
22	14h	Kontrolní bity TCP (6 bitů)

Tabulka 2.1: Unifikované hlavičky

čislo v UH	Protokol
01h	ICMP
06h	TCP
11h	UDP
3Ah	ICMPv6
jiný	Neznámý protokol

Tabulka 2.2: Protokol transportní vrstvy

až 3.2 Gb/s při synchronizační frekvenci 100Mhz. Výsledkem vyhledávání je vektor, který určuje jaké řetězce byly nalezeny. Nad tímto vektorem a vektorem výsledků klasifikace z klasifikátoru je pak provedena operace logický součin a následovně logický součet. Výsledkem je zjištění, zda paket splnil některé pravidlo a je tedy podezřelý. [17]

2.3.2 Rozhraní

Jednotka má dvě vstupní rozhraní a jedno rozhraní výstupní.

Vstupní rozhraní je tvořeno 32 bitovou sběrnicí FrameLink. Přenášený rámec má 3 části. V první části, která má velikost 32 bitů obsahuje identifikaci interface, který přijal paket a velikost paketu v bytech. Ve druhé části jsou uloženy hlavičky síťových protokolů. V poslední části jsou uložena užitečná data paketu, ve které jsou následovně hledány řetězce.

Rozhraní od klasifikátoru je složeno z vektoru CLASIFICATION o 5000 prvcích (tento počet je možno měnit při překladu), jehož obsah je validní při aktivním signálu CLASIFICATION_VLD a je potvrzováno jednotkou nastavením signálu CLASIFICATION_ACK. Tento vektor obsahuje informaci o tom, která pravidla byla splněna paketem při klasifikaci.

Výstupní rozhraní je tvořeno signálem PTRN_MATCH_MATCH, který je aktivní pokud je paket podezřelý. Hodnota signálu je platná pokud je aktivní příznak ukončení klasifikace PTRN_MATCH_EOP.

Posledním rozhranním je rozhraní typu MI32, které slouží především pro sběr informací o stavu komponenty.

2.4 Klasifikátor

2.4.1 Popis

Klasifikátor slouží ke klasifikaci paketů podle pěti příznaků. Jedná se o protokol, zdrojovou IP adresu, cílovou IP adresu, zdrojový port a cílový port. Část kódu klasifikátoru je generována z pravidel snorta, tudíž je třeba při změně pravidel přeložit a nahrát do FPGA nový design. Jednotka používá CAM paměť na čipu a pole prvků typu logický součin, které je využito pro klasifikaci podle protokolu. Díky použití paměti CAM je možné provádět klasifikaci podle všech pravidel najednou. [29]

2.4.2 Rozhraní

Jednotka má jedno vstupní a jedno výstupní rozhraní.

Vstupním rozhraním je 32 bitový protokol FrameLink obsahující unifikované hlavičky. Pro popis unifikovaných hlaviček viz. 2.2.2.

Výstupem klasifikace je vektor RESULT o 5000 prvcích (tento počet je možno měnit při překladu), který obsahuje informace o tom, která pravidla byla splněna. Obsah tohoto vektoru je platný pokud je signál RESULT_VLD aktivní. Přijetí dat musí být potvrzeno odebírající jednotkou nastavením signálu RESULT_ACK do aktivního stavu.

Dále komponenta obsahuje dvě rozhranní typu MI32. Jedno slouží pro zjišťování informací o stavu komponenty a druhé pro nahrávání obsahu paměti CAM.

Kapitola 3

Rozšíření o podporu IPv6

3.1 Úvod

S postupným rozšiřováním protokolu IPv6 a vydáním nové verze systému Snort (verze 2.8 viz. [18]) vznikla nutnost upravit stavající klasifikátor. Klasifikátor původního IDS podporoval pouze protokol IPv4.[29]

3.2 IPv6 z pohledu klasifikátoru

Protokol IPv6 přinesl mnohé novinky. Z pohledu klasifikátoru je nejdůležitější změnou změna adresování. Na rozdíl od IPv4, kde jsou adresy 32 bitové, zavádí IPv6 adresy 128 bitové [25]. Další důležitou změnou je to, že došlo ke změně protokolu ICMP. Nová verze ICMPv6 sloučila služby poskytované dříve protokoly ICMP, ARP, IGMP a přidala k nim další služby jako bezestavové přidělování adres, atd. [21] Současně nastává problém v tom, že protokoly IPv4 a IPv6 mohou na síti koexistovat.

Z pohledu klasifikátoru je důležité umístění pole verze v hlavičce na stejném místě, což umožňuje lehce odlišit verzi protokolu IP. Kromě polí zdrojové a cílové adresy nejsou ostatní pole pro klasifikátor zajímavé.

3.3 Adresování v IPv6

IPv6 přináší 128 bitové adresy, které přináší 2^128 možných adres. Tímto byl vyřešen nejzávažnější problém IPv4 a to nedostatek adres. Adresy IPv6 se skládají ze 64 bitového prefixu sítě a 64 bitové části pro identifikaci konkrétního zařízení. Tato část adresy může být generována na základě MAC adresy, či přiřazena. Z důvodu zachování anonymity se vyvinuly systémy pomocí nichž se negeneruje tato část z unikátní MAC adresy, ale je použita metoda generování náhodných řetězců viz. [25]. Prefixy se zapisují stejně jako u CIDR jako lomítko následované délkou prefixu.

Adresy IPv6 se zapisují jako osm čtyř bytových hexadecimálních číslic oddělených dvojtečkou. Uvozující nuly každého bloku je možné vynechat. Jednu skupinu více nulových bloků je možné nahradit dvěmi dvojtečkami za sebou [25]. Příklad:

```
2001:0db8:0000:0000:0000:00aa:0000:0bed -> 2001:db8::aa:0:bed
```

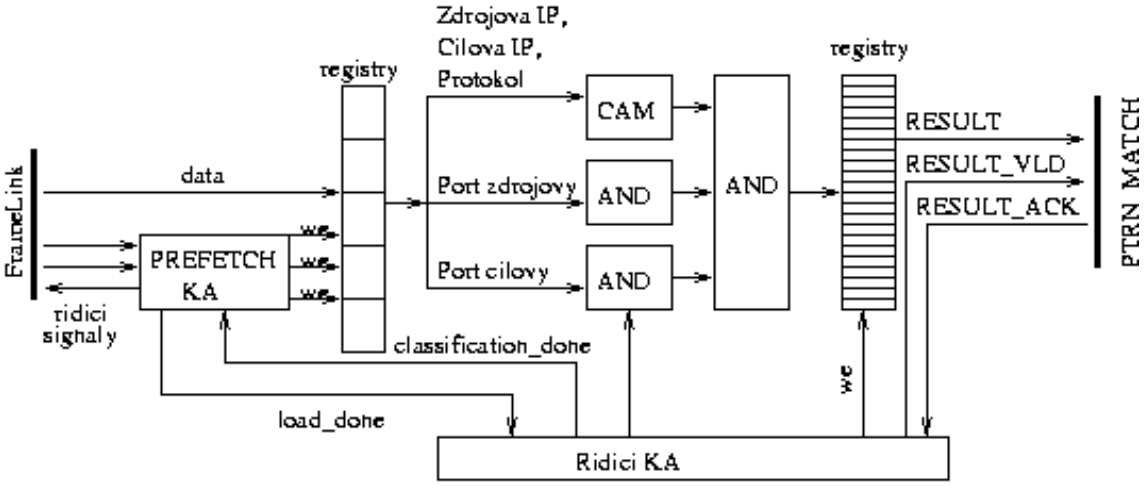
IPv6 se dělí na adresy typu unicast, multicast a anycast [25]. Toto rozlišení však není z hlediska klasifikátoru podstatné. Podstatnější je, že prefix ::ffff:0:0/96 se používá pro

mapování IPv4 adres do IPv6 adresového prostoru [24]. Toto je běžně využíváno pro IP stack, který tak reprezentuje IPv4 adresy aplikacím napsaným pro IPv6. Díky možnosti použití tohoto prefixu je mimo jiné možná dobrá koexistence IPv4 a IPv6 na jedné síti. Tento způsob mapování IPv4 adres na IPv6 adresy je využit v klasifikátoru s podporou IPv6. Tento způsob transformace má určitá omezení. Tato omezení se týkají toho, že některá pole hlavičky IPv4 nelze transformovat na pole hlavičky IPv6 [24].

3.4 Návrh a implementace

Po analýze současné struktury klasifikátoru jsem došel k závěru, že nejlepším řešením bude upravit paměť typu CAM tak, aby místo adres IPv4 obsahovala adresy IPv6 s využitím mapování adres IPv4 na adresy IPv6. U klasifikátoru nezáleží na omezeních, která vyplývají z tohoto způsobu transformace adres. Paměť CAM byla rozšířena z 68 bitů (4 bity protokol + 2 x 32 bitů adresy) na 260 bitů (4 bity protokol + 2 x 128 bitů adresy) na položku. Dále je nutné změnit konečný automat jednotky prefetch, která zajišťuje načítání z unifikovaných hlaviček ze vstupního FrameLinkového rozhraní do registrů jednotky klasifikátoru. Tato změna spočívá v přidání podpory zpracování unifikovaných hlaviček vztahujících se k protokolu IPv6, tj. adres. V jednotce klasifikátoru byly rozšířeny registry pro uložení IP adres na 128 bitů. Také byla přidána možnost resetovat tyto registry po zkončení zpracovávání unifikovaných hlaviček. Registry se resetují na hodnotu odpovídající binární reprezentaci prefixu ::ffff:0:0/96, což umožňuje snadné nahrávání IPv4 adres, které změní pouze odpovídajících 32 bitů.

Rozšířením délky jedné položky paměti CAM na 260 bitů došlo k nárůstu velikosti zabrané plochy na čipu FPGA pamětí CAM. U 32 položek v paměti CAM byla zabrána plocha cca. 6% čipu VirtexII-Pro na kartě COMBO6X. Tento problém však není tak vážný, jak by se mohlo zdát, neboť pravidla jsou často postavená tak, že více pravidel má stejný protokol a zdrojovou i cílovou adresu, tudíž může sdílet stejné položku paměti CAM. Schéma klasifikátoru je možné najít na obr. 3.1



Obrázek 3.1: Schéma klasifikátoru

Kapitola 4

Verifikační metodologie

4.1 Úvod

Cílem verifikace není hledání chyb, nýbrž ověření zda se design chová podle zadaných specifikací. Proto je vhodné, aby práce na verifikaci probíhaly souběžně s procesem tvorby designu. Dále je vhodné, aby pokud je to možné nedělal verifikaci a design jeden člověk, ale více lidí poněvadž platí, že více lidí bude mít na specifikaci více různých pohledů. Tato kapitola vychází především z [5].

4.2 Místa výskytu chyb

Chyby se mohou vyskytovat na různých úrovních. Na úrovni jednotlivých bloků se jedná o chyby vytvořené jedním člověkem, tudíž snadno odhalitelné řízenými testy. Další úrovní je úroveň na styku více komponent. Tady vznikají například chyby vyplývající z rozdílného pochopení stejné specifikace [5].

Jelikož testování jednotlivých komponent je rychlejší než testování složitého celku, je doporučeno postupovat při testování tak, že se nejdříve otestují jednotlivé komponenty a až pak výsledný celek. [5] S rostoucí složitostí designu roste i obtížnost jeho verifikace.

4.3 Metody Verifikace

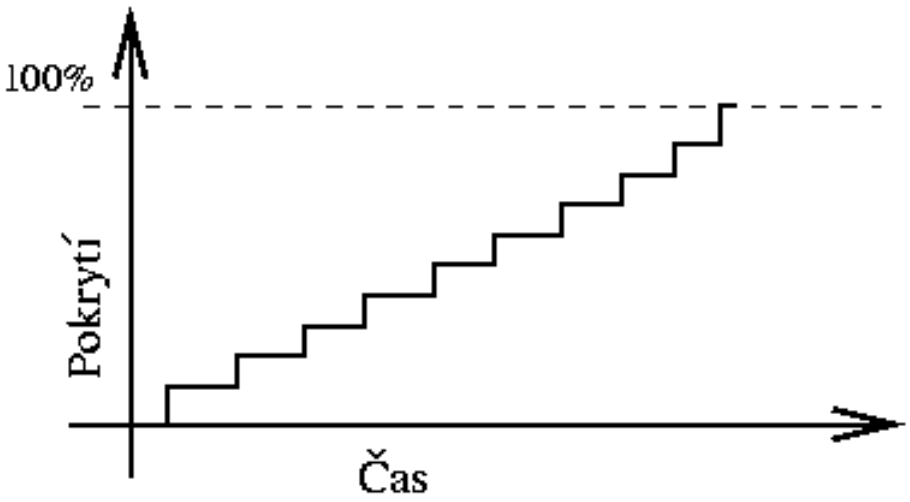
4.3.1 Formální verifikace

Formální verifikace provádí verifikaci designu formálně tj. matematicky. Nezbytným předpokladem pro provedení formální verifikace je existence formální specifikace zapsané formální logikou. Jediným plně automatizovatelným typem formální verifikace je model checking, tudíž musí být formální specifikace zapsána formulemi temporální logiky. Výhodou formální verifikace je její úplnost, tudíž je vhodná pro verifikaci kriticky důležitých designů. Nevýhody formální verifikace jsou nutnost znalosti temporální logiky pro vytvoření formální specifikace a hluboké pochopení problematiky hardware pro specifikaci verifikačního prostředí. Výše uvedené nevýhody způsobují výrazný pokles rychlosti verifikace designu, takže nestačí rychlosti vývoje designu. [4]

4.3.2 Řízené testování

Jedná se o tradiční způsob verifikace prostřednictvím testbenchů. Verifikátor si přečte specifikaci designu, na základě které vytvoří verifikační plán. Verifikační plán obsahuje sadu testů. Každý z nich se zaměřuje na otestování určité části vlastností. Na základě těchto plánů se provede návrh testovacích vektorů, jenž se vyšlou do testovaného designu. Výstupy z jednotky jsou pak ručně porovnány s předpokládanými výsledky. Pokud test proběhne správně, implementuje další test v pořadí. [5]

Výhodou tohoto přístupu je stálý postup verifikace vpřed a brzké počáteční výsledky (viz obrázek 4.1). Nevýhodou je naopak dlouhá doba než se verifikace dokončí a tudíž náročnost na lidské zdroje. Další nevýhodou je, že testovací vektory navrhuje člověk a tudíž se zaměřuje na oblasti, kde si myslí, že by chyby mohly být. Tudíž nemusí testy obsáhnou všechny chyby (viz obrázek 4.2). Také platí, že zdvojnásobíme-li složitost designu, pak se doba potřebná k verifikaci také zdvojnásobí. [5]



Obrázek 4.1: Graf postupu řízeného testování. (podle [5])

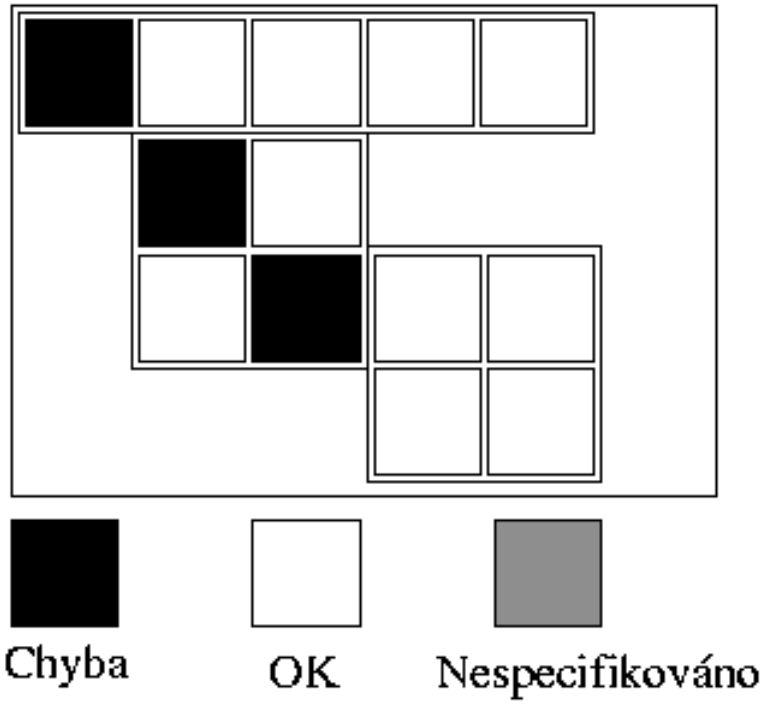
4.3.3 Constrined-random test

Popis

Testbench by měl provádět tyto základní činnosti při verifikaci (podle [5]):

- Vytvářet stimuly.
- Aplikovat tyto stimuly na testovaný design.
- Získávat reakce na tyto stimuly.
- Kontrolovat správnost reakcí.
- Měřit pokrok verifikace oproti cílům verifikace.

Řízené testování provádí automaticky pouze položky dvě a tři výše uvedeného seznamu. Zbylé se provádí manuálně. Časová náročnost tohoto zpracování vedla k vytvoření metodologie, která by umožňovala provádět tyto činnosti automaticky.



Obrázek 4.2: Pokrytí prostoru jednotlivými testy. (podle [5])

Tato metodologie staví na následujících principech (dle [5]):

Náhodné stimuly: Generováním náhodných stimulů můžeme otestovat i chyby, o kterých bychom nikdy nepředpokládali, že mohou vzniknout (viz. obr) .

Functional coverage: Jelikož používáme náhodné stimuly, musíme měřit pokrok verifikace oproti cílům verifikace.

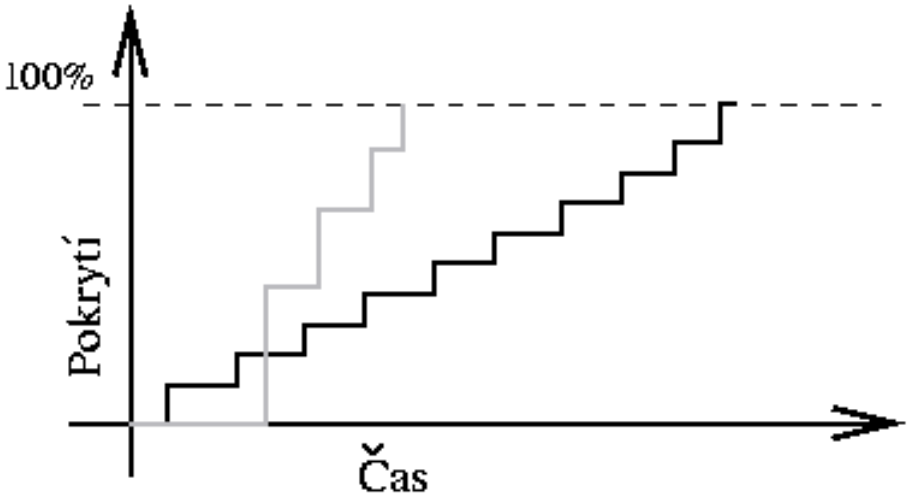
Vrstevný testbench s použitím transakcí: Rozdělením testbenche do vrstev a jednotlivých spolu komunikujících modů docílíme toho, že se nám bude lépe řešit jeho vytvoření. Testbench totiž na rozdíl od řízeného testování musí provádět i automatickou kontrolu výsledků, generování náhodných stimulů, apod.

Společný testbench pro všechny testy: Rozdělením testbeche na jednotlivé moduly a vhodným návrhem docílíme toho, že ho nebudeme muset s každým testem přepisovat.

Kód pro jednotlivé testy oddělen od společného testbenche: Kód specifický pro jednotlivé testy by měl být oddělen od společného tesbenche, aby ho zbytečně nekomplikoval.

Vytvoření této infrastruktury zabere delší dobu než v případě řízeného testování, ale následný proces verifikace je rychlejší (viz. obrázek 4.3). Náhodné testy mohou také odhalit chyby a nespecifikované chování, které by nebyly odhaleny řízenými testy (viz obrázek 4.4).

Při vlastní verifikaci je vhodné postupovat se zakomponováním zpětné vazby do procesu verifikace (viz obrázek 4.5). Na začátku se začne s testy se základními omezeními náhodného generování, ty se nechají několikrát proběhnout s různým nastavením seedu



Obrázek 4.3: Graf postupu náhodného testování testování (šedě) oproti řízenému testování (černě). (podle [5])

generátoru pseudonáhodných čísel, přičemž se měří fuctional coverage. Poté se určí chyby, udělají se minimální změny kódu, případně se upraví omezení a test se opakuje, dokud nedosáhneme požadovaných výsledků. Řízené testy by se měly používat pouze pro testování vlastností, které by bylo náhodnými testy obtížné dosáhnout. Zakomponování této zpětné vazby umožňuje zrychlit proces verifikace (viz obrázek 4.6). [5]

Komponenty vrstevného testbenche

Vrstevný testbench se skládá z několika vrstev, které seskupují logicky k sobě patřící komponenty (podle [5]) viz obrázek 4.7. Vrstvy command, functional a scenario spolu tvoří prostředí, které simuluje okolní HW komponenty. Tyto vrstvy by měli být vytvářeny současně s designem DUT již od začátku vývoje. V závislosti na složitosti verifikovaného designu je možné některé časti vynechat nebo je sloučit s jinými. Více o metodologii se můžete dočíst v [1] [11].

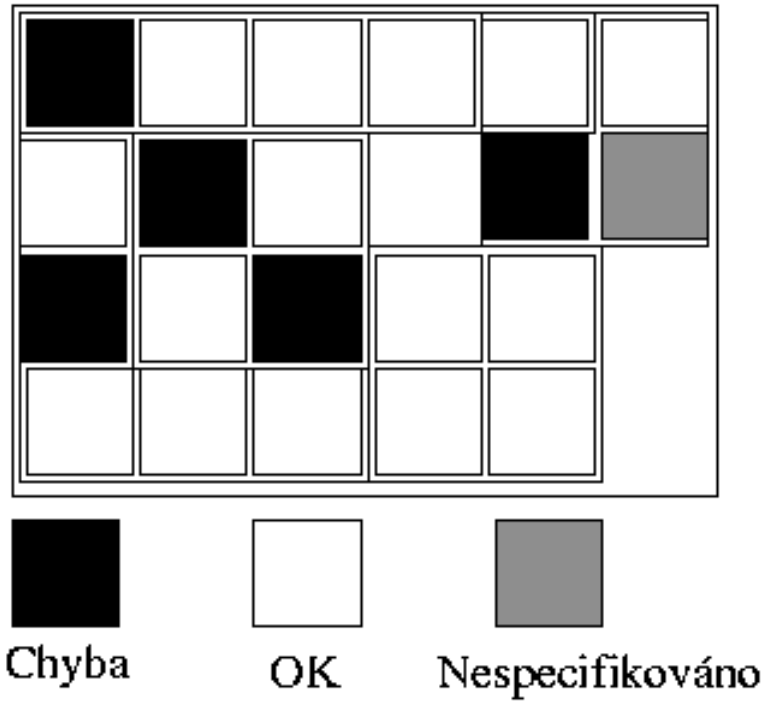
Vrstva signal V této vrstvě se nachází samotný testovaný design. Je označen DUT z anglického design under test. Tato jednotka je připojena ke zbytku testbenche signály.

Vrstva command V následující vrstvě se nachází několik jednotek. Jedná se o driver, monitor a assertions.

Driver vytváří na základě jednotlivých příkazů od agenta signály pro DUT. Monitor provádí činnost opačnou, tj. přijmá výstupní signály DUT a vytváří z nich jednotlivé příkazy. Assertions slouží pro monitorování stavu jednotlivých signálů DUT.

Vrstva functional V této vrstvě se nachází jednotky agent, scoreboard a checker.

Agent rozděluje transakce na jednotlivé příkazy, které zasílá driveru a scoreboardu. Scorbord na základě přijatých příkazů predikuje očekávaný výstup, který je pak srovnán jednotkou checker s výstupem DUT.

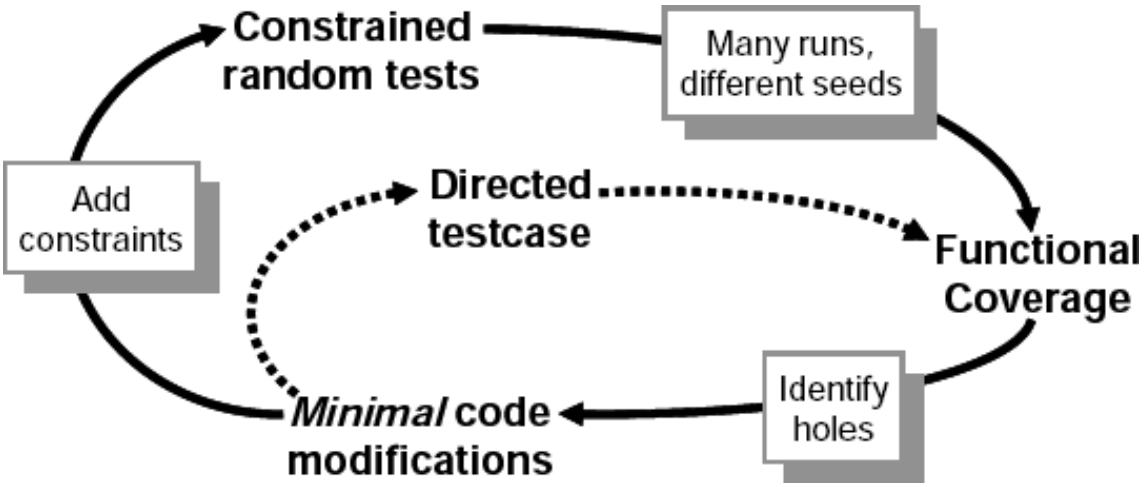


Obrázek 4.4: Pokrytí prostoru jednotlivými náhodnými testy. (podle [5])

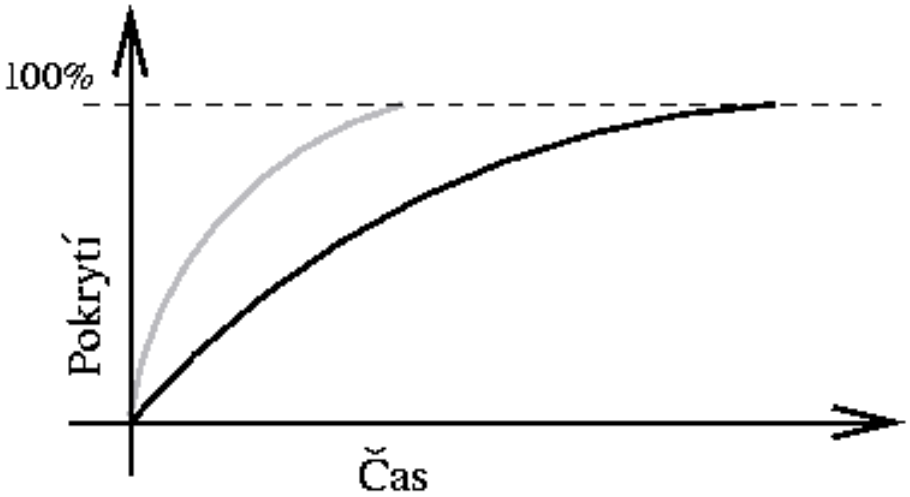
Vrstva scenario Úlohou generátoru v této vrstvě je řídit vrstvu functional. Provádí randomizaci transakcí, na základě scénáře. Scénářem může být například přenos paketu. Generátor je většinou implementován s použitím návrhového vzoru factory.

Vrstva test a functional coverage Test řídí celý proces verifikace. Nastavuje například parametry generátoru.

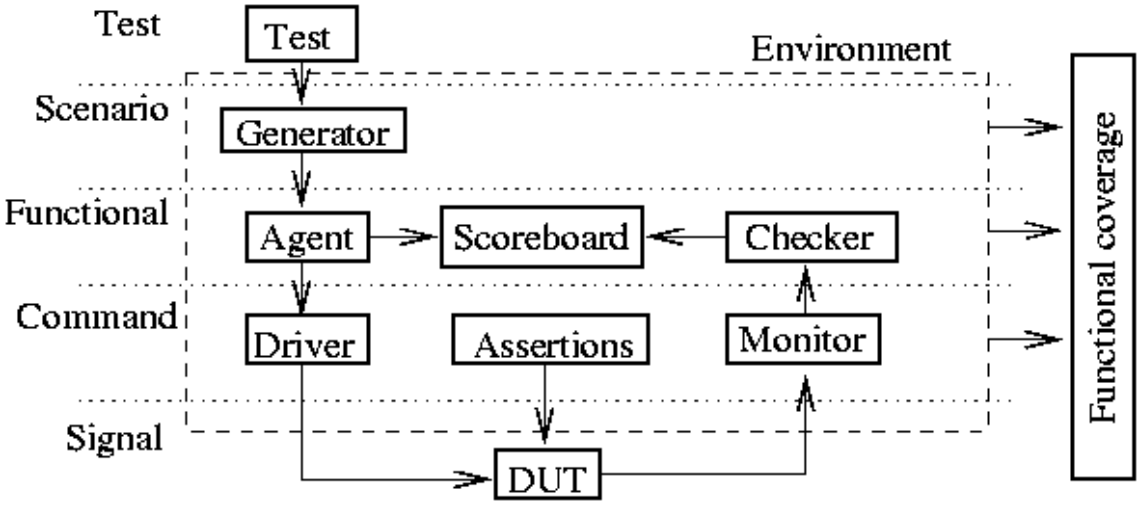
Functional coverage slouží k měření postupu všech testů a bývá často měněna v průběhu vývoje. Z tohoto důvodu není součástí prostředí.



Obrázek 4.5: Zpětná vazba v procesu verifikace. (převzato z [5] - Figure 1-5)



Obrázek 4.6: Význam zpětné vazby v procesu verifikace. Černě bez zpětné vazby, šedě se zpětnou vazbou. (podle z [5])



Obrázek 4.7: Vrstvy testbenche. (podle z [5])

Kapitola 5

SystemVerilog

5.1 Úvod

Historie jazyka SystemVerilog se začíná psát roce 2002, kdy byl organizaci Accelera darován jazyk Superlog, což bylo rozšíření jazyka Verilog. Spolu s jazykem OpenVera, který poskytnul verifikační část, vytvořil základ jazyka SystemVerilog. V roce 2003 byl tento jazyk standardizován organizací Accelera. V roce 2005 se stal jazykem standardizovaným organizací IEEE. [26]

SystemVerilog v sobě kombinuje vlastnosti jak jazyků pro popis hardware, tak jazyků pro verifikaci hardware. [5]

V následujících částech budou popsány důležité konstrukce a rysy jazyka. Obsah těchto částí byl vytvořen na základě [5] [19] [10].

5.2 Datové typy

5.2.1 Základní datové typy

SystemVerilog podporuje velké množství základních datových typů. Tyto typy můžeme rozdělit na typy čtyřhodnotové a dvouhodnotové. Mezi čtyřhodnotové typy patří typ logic a wire. Mezi typy dvouhodnotové patří bit, byte, shortint, int a long int. Typ bit zaujímá mezi ostatními zvláštní postavení, neboť je vždy bez znaménka a umožňuje vytvářet libovolně velké proměnné. Ostatní typy mají pevně danou velikost a jsou standardně znaménkové, přičemž se mohou přepnout na bezznaménkové pomocí klíčového slova unsigned. Při používaní znaménkových typů je třeba brát zřetel na to, že mohou způsobovat problémy při randomizaci. Všechny tyto typy je možné indexovat i po bitech.

5.2.2 Pole

SystemVerilog podporuje multidimenzionální pole jak pevné velikosti, tak proměnné velikosti. SystemVerilog umožňuje deklarovat pole jako packed a unpacked, přičemž jedno pole může být obou dvou typů. Pole typu unpacked se vytvoří tak, že se dimenze pole zapíší až za název pole. Typ unpacked značí, že prvky pole budou ukládány tak, že velikost každého prvku bude zaokrouhlena na násobky 32-bitů. To například u pole jednobitových hodnot typu bit představuje plýtvání paměťovým prostorem. Pro lepší využití může být pole deklarováno jako typ packed. Při použití tohoto typu jsou data ukládána hned za sebou bez volných míst. Toto znamenná nejen, že paměť bude využívána efektivněji, ale umožňuje to

pracovat s celým polem místo s jeho jednotlivými prvky. Pole typu packed mají to omezení, že mohou být pouze fixní velikosti, tj. nemohou být tohoto typu dynamická pole, asociativní pole, nebo fronty.

Pole je možné kopírovat a porovnávat jako celek. Porovnávání je omezeno jen na rovnost a nerovnost. I při práci s polem je možné adresovat jednotlivé bity prvků pole.

5.2.3 Fronta

SystemVerilog zavedl užitečný datový typ fronta. Fronta je deklarována podobně jako pole, ale v deklaraci mezí má uveden znak $. Datový typ fronta umožňuje vkládání a mazání kdekoliv ve svém obsahu, přičemž práce se začátkem a koncem fronty je nejrychlejší. Vkládání do středu fronty je nejpomalejší, neboť je třeba přesunout až půlku obsahu fronty.

5.2.4 Asociativní pole

Asociativní pole umožňuje vytvářet rozsáhlá pole, která však nezabírají velkou oblast paměti. Toto pole totiž obsahuje pouze prvky, které jsme do něj zapsali. Je vhodné například pro simulaci velké paměti, ve které pracujeme pouze s několika rozptýlenými položkami.

Asociativní pole může být indexováno buď číslem, nebo řetězcem. Pokud je indexováno číslem, pak se musí deklarovat s použitím *. Indexování řetězcem se při deklaraci zadá klíčovým slovem string.

Asociativní pole poskytuje několik funkcí pro práci s indexy pole a s obsahem pole. Jedná se zejména o získání počátečního a následujícího indexu a o smazání prvku asociativního pole. Také je použitelná funkce exists, která zjišťuje zda zadaný prvek v poli existuje.

5.2.5 Metody polí

S poli typu unpacked (což mohou být fixní, dynamická a asosiativní pole a fronty) mohou být použity metody polí. Tyto metody zahrnují operace od součtu prvků v poli po jejich seřazení. Metody můžeme rozdělit na několik skupin.

Jednou ze skupin jsou metody operující nad prvky pole a vracející skalární hodnotu. Do této skupiny patří metody jako sum, product, and, or, xor. Například metoda sum provede součet všech prvků pole.

Další skupinou jsou metody, které vyhledávají prvky pole podle nějakého kritéria. Tyto metody vrací výsledek ve formě fronty. Do této skupiny patří například metody min a max čí metoda find.

Řazení polí lze provádět pomocí metod polí pouze pro pole jednorozměrná.

5.2.6 Uživatelsky definované typy

SystemVerilog umožňuje vytvářet uživatelsky definovanné typy. Podporuje typy struct a union, přičemž u struct může být použito klíčové slovo packed, což přináší obdobné vlastnosti jaké jsou popsány výše u polí. Jinak se chovají podobně jako v jazyce C.

Výčtové typy je možno také vytvořit v jazyce SystemVerilog. Proměnné výčtového typu umožňují použít několik metod pro práci s nimi. Například metoda name vrátí jméno položky výčtového typu, nebo metody first, last, next nebo prev umožňují průchod přes hodnoty výčtového typu.

5.2.7 Řetězce

SystemVerilog umožňuje práci se řetězci. Řetězec se chová jako dynamické pole jehož jednotlivé prvky jsou typu byte. Řetězec umožňuje používat pro práci s jeho obsahem metody jako getc, putc, toupper, tolower, substr apod.

5.3 Objektově orientované programování

Jeden z největších přínosů SystemVerilogu je podpora objektově orientovaného programování. Díky podpoře objektově orientovaného programování je možné psát kód zejména pro účely verifikace mnohem efektivněji. Také se lépe dodržuje členění tesbenche do jednotlivých částí.

Objekty jsou vytvářeny voláním konstruktoru - metody new. Jelikož SystemVerilog používá garbage collector, existuje objekt do té doby dokud na něj existuje nějaká reference. Poté je odstraněn. Pokud chceme odstranit objekt nastavíme handle tohoto objektu na null, čímž odstraníme referenci.

SystemVerilog podporuje statické atributy tříd. Viditelnost atributů tříd je možné nastavit na public nebo private, přičemž defaultní nastavení je public.

SystemVerilogem je podporována jednoduchá dědičnost. Při použití klíčového slova virtual u metod, je možné implementovat polymorfismus.

5.4 Randomizace

5.4.1 Úvod

Při implementaci CRT se používá automatické generování testcases. CRT se obecně skládá ze dvou částí: Testovacího kódu, který vytváří náhodný vstup pro DUT a z generátoru pseudonáhodných čísel. Při změně seedu tohoto generátoru se změní chování celého systému.

Randomizovat můžeme širokou škálu stimulů. Nejjednodušší se může zdát randomizace dat. Toto však není dostačující, poněvadž by jsme ověřili pouze datovou cestu. Pro kompletní ověření jednotky je třeba randomizovat také kontrolní cestu. Randomizovat můžeme širokou škálu vstupů jednotky, například podle [[5]]:

Konfiguraci zařízení: Je třeba vyzkoušet co nejvíce možných konfigurací zařízení. (Například počet kanálů, ...)

Konfiguraci prostředí: Je nutné nastavit prostředí testbenche tak, aby co nejlépe simuloval možná reálná zapojení. (Například počet rozhraní,...)

Vstupní data: Randomizace vstupních dat (FrameLinkový rámec,...)

Zapouzdřená vstupní data: Vstupní stimuly se mohou skládat z různých úrovní, které je nutné správně randomizovat (Například IP paket do Ethernetového rámce,...)

Vyjímky a chyby protokol: Je nutné zjistit co se stane, když dojde k nějaké chybě ve vstupním protokolu, proto je vhodné náhodně vkládat v náhodných intervalech do protokolu chyby.

Zpoždění: Protokoly často počítají s tím, že nějaká data přijdou v určitém intervalu po žádosti o ně. Některé jednotky deklarují ochotu komunikovat s druhou jednotkou.

Obecně se dá říci, že čím větší množství stimulů bude náhodné, tím větší bude pravděpodobnost nalezení chyb.

5.4.2 Prostředky SystemVerilogu pro randomizaci

Randomizace bývá nejúčinější tehdy, když se využije ve spolupráci s objektově orientovaným programováním, kdy seskupíme spřízněné proměnné do tříd. Randomizaci nějakého objektu spustíme zavoláním metody randomize. Pokud při randomizaci dojde k nějakému problému vrací tato funkce 0.

Rand a randc

Pomocí klíčových slov rand a randc je možné nastavit, že určitá proměnná nebo atribut bude randomizována. Klíčové slovo rand značí, že bude proměnné přiřazena náhodná hodnota, přičemž se tato hodnota může kdykoliv opakovat. Použitím randc docílíme toho, že budou vygenerovány všechny hodnoty, než se začnou náhodné hodnoty opakovat.

Omezení generovaných hodnot

Při generování náhodných čísel často potřebujeme rozsah generovanných hodnot omezit. SystemVerilog pro tyto účely nabízí konstrukci uvozenou klíčovým slovem constraint:

```
constraint jmeno {
   a < 10;
   c > a;
}
```

V tomto příkladě byl vytvořen constraint jmeno, který říká, že atribut a musí být menší než 10 a atribut c musí být větší než a.

Omezení mohou nabývat složitých podob, je možné například nadeklarovat, že se hodnoty proměnné musí nacházet v určitých rozsazích, je možné stanovit závislost na jiných proměnných, u polí je možné například určit rozsahy dimenzí, rozsah hodnot prvků, je možné určit, jaký má být součet všech prvků v poli, apod. Pro další informace doporučuji nahlédnou do následujících publikací —[5] [19] [1].

Konstrukci constraint můžeme použít i s proměnnými, které nemají u sebe klíčové slovo rand, či randc. V tomto případě se při randomizaci zkontroluje zda se hodnoty této proměnné nachází v zadných mezích. Pokud se nenalézají je vrácena chyba.

Je nutné zdůraznit, že pravidla v konstrukci constraint jsou deklarativní, tudíž není zaručeno pořadí vyhodnocení. SystemVerilog umožňuje jednotlivé konstrukce constraint zapínat a vypínat za běhu a tudíž je možná další kontrola randomizace.

Metody pre_randomize a post_randomize

Někdy je nutné nastavit některé proměnné těsně před nebo těsně po randomizaci. Pro tento účel slouží metody pre_randomize a post_randomize, které jsou automaticky spuštěny před, nebo po randomizaci.

5.5 Vlákna a meziprocesová komunikace

5.5.1 Vlákna

Většina bloků v testbenchi pracuje paralelně a tudíž musí vytvářet vlákna. Pro tento účel poskytuje SystemVerilog konstrukce fork...join, fork...join_any, fork...join_none. Konstrukce fork...join čeká než skončí všechna vytvořenná vlákna, než se pokračuje, fork...join_any čeká na ukončení některého vytvořeného vlákna a fork...join_none nečeká na žádné vytvořené vlákno. Příkazy uvnitř této konstrukce se spouštějí paralelně, pokud chceme některé příkazy vykonat sekvenčně, musíme je uzavřít mezi begin a end.

Pro ukládání hodnot v konkurentních vláknech se musí používat automatické proměnné, jinak by si souběžně běžící vlákna přepisovala svá data. Automatická proměnná se vytvoří tak, že před typ proměnné umístíme klíčové slovo automatic. Každý proces pak používá svou kopii této proměnné.

Vytvořené vlákno můžeme zastavit pomocí příkazu disable.

5.5.2 Meziprocesová komunikace

Vytvořená vlákna mezi sebou potřebují komunikovat. Potřebují se synchronizovat či si mezi sebou předávat data. SystemVerilog poskytuje několik způsobů meziprocesové komunikace.

Události

Jednotlivá vlákna na sebe můžou čekat, tj. jedno vlákno čeká na to až druhé vlákno aktivuje událost. Událost se vytvoří pomocí klíčového slova event. Událost se vyvolá operátorem -¿. Na událost se čeká buď pomocí operátoru @, který je citlivý na hranu, nebo pomocí příkazu wait, který je citlivý na hladinu. Parametrem příkazu je výsledek metody události triggered.

Semafory

Semafory umožňují kontrolovat přístup ke zdrojům, například ke sběrnici. Je možné určit kolikrát může být zařízení používáno současně (v terminologii [5] počet klíčů) a vlákno může požádat o poskytnutí určitého počtu klíčů. Semafor se deklaruje pomocí klíčovího slova semaphore. Před použitím se musí pomocí funkce new alokovat celkový počet klíčů. Klíč lze získat pomocí metody semaforu get, klíč se vrací metodou put a je možné se pokusit získat klíč metodou try_get, která vrací 1 pokud existuje dostatečné množství klíčů a 0 pokud ne.

Mailbox

Pro přenos dat mezi jednotlivými vlákny je vhodné používat mailbox. Tento objekt system verilogu pracuje tak, že vysílající strana do něj vkládá data a přijímající strana si je vyzvedává. Mailbox může mít velikost buď neomezenou nebo omezenou. Pokud je velikost omezena, tak je vkládající strana zablokována dokud se neuvolní místo. Taktéž je čtecí strana zablokována pokud nejsou v mailboxu nějaká data. Mailbox je možné chápat jako frontu. Používání mailboxu umožňuje asynchronní komunikaci mezi dvěmi vlákny.

Mailbox se vytvoří voláním jeho konstruktoru new, jehož volitelným parametrem je jeho velikost. Poté lze do něj data vkládat pomocí metody put a vybírat pomocí get. Metodou peek je možné získat kopii prvku bez jeho vybrání. Metody try_put a try_get dělají to samé

co put a get, avšak při nemožnosti provedení požadované akce se nezablokují, nýbrž vrátí 0. V případě úspěchu vrací 1.

Je důležité si uvědomit, že do mailboxu se dají umístit pouze skalární typy (například int, handle, logický typ libovolné velikosti,... - ne však pole apod.)

5.6 Function coverage

Function coverage slouží pro měření, které vlastnosti designu byly vyzkoušeny během testů. Můžeme například měřit zda jsme vyzkoušeli všechny kombinace řídících signálů na sběrnici, stavy designu, apod. Při analýze výsledků verifikace je vhodné používat zpětnovazební smyčku pro určení jaké změny mají být provedeny za účelem dosažení 100% pokrytí (viz obrázek 4.5). Zvýšení pokrytí můžeme dosáhnout spuštěním existujících testů s různými seedy generátoru náhodných čísel, nebo můžeme upravit omezení při generování náhodných hodnot. Prostředky pro zjišťování functional coverage v SystemVerilogu jsou popsány například v [5].

5.6.1 Code coverage

Code coverage slouží k získání informací o tom, které řádky kódu byly použity, kterými stavy prošel konečný automat, apod. Nejedná se o rys jazyka SystemVerilog, ale o vlastnost daného simulačního nástroje. Zjistíme jím, zda náš test otestoval všechny řádky kódu, ale nezjistíme chyby v designu oproti specifikaci (například zapomenutou reakci na nějaký signál).

5.6.2 Assertion coverage

Slouží ke zjištění zda byly pokryty všechny assertions. Assertion kontroluje vztahy mezi jednotlivými signály. Může být například použit pro kontrolu zda signály přišly ve správném pořadí.

5.6.3 Zásady pro použití function coverage

Při používání code coverage je vhodné dodržovat určitá pravidla (podle [5]):

Získávejte informace ne data: Není potřeba získávat data o dosažení všech hodnot například 32-bitového čítače. Zaměříme se na získávání informací o hraničních podmínkách (Například zajímá nás jak se čítač chová v okolí nuly a maximální hodnoty, zbylé hodnoty shrneme do jedné kategorie). Získáme tak mnohem užitečnější a přehlednější zprávu o dosažených výsledcích.

Získávejte pouze potřebné informace: Jelikož je získávání informací o pokrytí proces náročný jak na výpočetní čas tak na paměť, je nutné získávat pouze data, která pak použijeme.

Kompletnost: Při určení zda jsme dosáhli dostatečné kvality otestování designu použijeme functional coverage i code coverage. Pokud jsou obě pokrytí splněna je možné říct, že jsme vytvořili test, který je na hledání chyb dostatečně účinný.

Kapitola 6

Verifikace systému pro detekci nežádoucího provozu

Při verifikaci jsem se rozhodl postupovat tak, že zverifikuji nejdříve klíčové komponenty designu jako jsou HFE_C, Classifier a Ptrn_match. Poté se může provést verifikace celého designu. Ostatní komponenty patří mezi obecné FrameLinkové komponenty, které jsou verifikovány odděleně.

6.1 Návrh

6.1.1 Systém

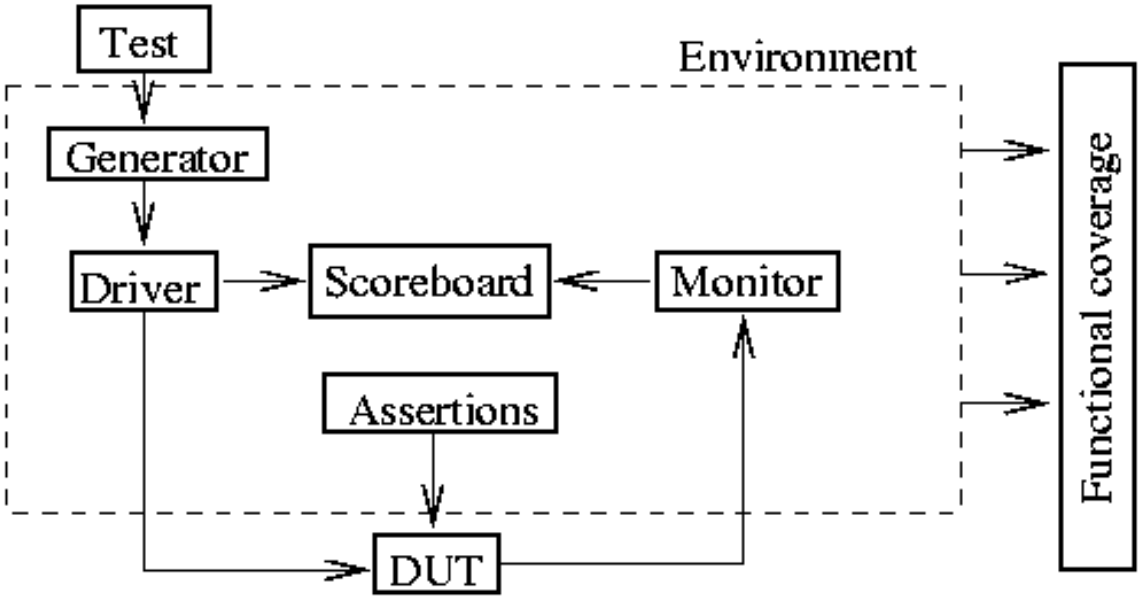
Systém je založen na jednotkách vyvinutých pro účely verifikace FrameLinkových komponent. Jedná se o základní komponenty jako je Framelinková transakce, driver, monitor [34]. Při verifikacích se používá metodologie vrstevnatého testbenche, přičemž jsou sloučeny jednotky agent s monitorem a checker se scoreboardem (viz obrázek 6.1). Tyto jednotky jsou sloučeny proto, že spolu funkčně souvisí a další dělení by zvyšovalo složitost. Monitor je rozdělen na vlastní monitor, který zpracovává signály a na responder, který generuje signály sběrnice.

Základní schéma systému pro verifikaci jednotlivých komponent je podobné, liší se podle počtu a typu rozhraní testovaného designu. Schémata jednotlivých propojení komponent je možné nalézt na 6.2, 6.3 a 6.4 . Rozhraní výše uvedených komponent je možné nalézt výše v 2.

Při verifikaci jsem se soustředil na verifikaci kritických částí jednotlivých komponent, tudíž jsem vynechal rozhraní typu MI32 u komponent Classifier a Ptrn_match. Tato rozhraní slouží především k zjišťování stavu komponent a proto nejsou pro funkci komponenty kritické. Toto neplatí pro rozhraní komponenty Classifier pro nahrávání obsahu CAM, avšak funkčnost tohoto rozhranní již bylo ověřena.

Systém je navržen tak, aby bylo možné velice jednoduše provést verifikaci systému pro detekci nežádoucího provozu jako celku. Stačí pouze vyskládat komponenty a implementovat model chování designu.

Data se mezi generátorem a driverem předávají pomocí mailboxu, poněvadž tyto jednotky mohou fungovat asynchronně. Driver a monitor se scoreboardem komunikují pomocí tzv. callback funkcí.



Obrázek 6.1: Modifikovaná metodologie vrstevnatého testbenche.

6.1.2 Paketový generátor

Srdcem celého systému je paketový generátor, který je vytvořen podle návrhového vzoru factory. Při inicializaci generátoru je vložena paketová transakce a síťové protokoly. Při vkládání síťových protokolů se také zadává pravděpodobnost s jakou budou generovány. Pravděpodobnost se zadává jako celé číslo určující podíl na celku. (Například je vložen protokol se zadanou pravděpodobností 1 a protokol s pravděpodobností 2. Pak se bude první protokol generovat s pravděpodobností 1/3 a druhý s pravděpodobností 2/3). Po spuštění generátoru se postupuje podle vývojového diagramu 6.5.

Generátoru lze nastavit minimální a maximální velikost vygenerovaného paketu.

Paketová transakce je zděděna od FrameLinkové transakce, ke které přidává další funkcionalitu a která přenáší objekty protokolů pro jejich zpracování ve scoreboardu. Data jsou přenášena jak ve formě FrameLinkového rámce, tak ve formě objeků jednotlivých protokolů proto, aby bylo možné ve scoreboardu jednodušeji implementovat model chování testovací jednotky a tak predikovat výstupy testované jednotky.

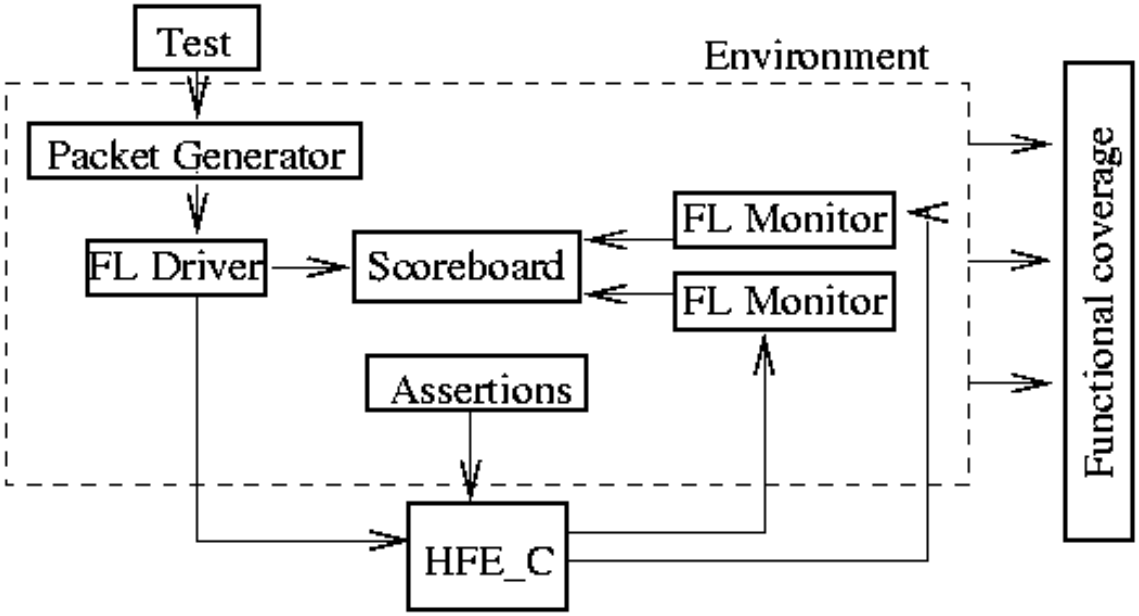
Objekty protokolů jsou přenášeny ve formě oboustranně zřetězeného seznamu, což umožňuje komunikaci objektů jednotlivých protokolů mezi sebou.

6.1.3 Podporované síťové protokoly

Podporovány jsou vybrané nejpoužívanější síťové protokoly s přihlédnutím k protokolům z nichž se extrahují hlavičky s pomocí HFE_C (v konfiguraci pro IDS). Podporované protokoly jsou (podle jednotlivých vrstev od nejnižší po nejvyšší) v tabulce 6.1.

Při návrhu bylo dbáno na to, aby bylo možné jednoduše přidávat další síťové protokoly. Při vytváření dalšího protokolu stačí implementovat třídu popisující protokol a u protokolů nižší vrstvy nastavit, že nově implementovaný protokol je použitelný s tímto protokolem nižší vrstvy.

Na obrázku 6.6 je zobrazeno které protokoly vyšší vrstvy mohou následovat za protokoly



Obrázek 6.2: Schéma pro verifikaci komponenty HFE_C.

nižší vrstvy. Pokud je zapnuta možnost zapouzdření protokolů pak se použijí i spojení v rámci jedné vrstvy či směřující do nižší vrstvy.

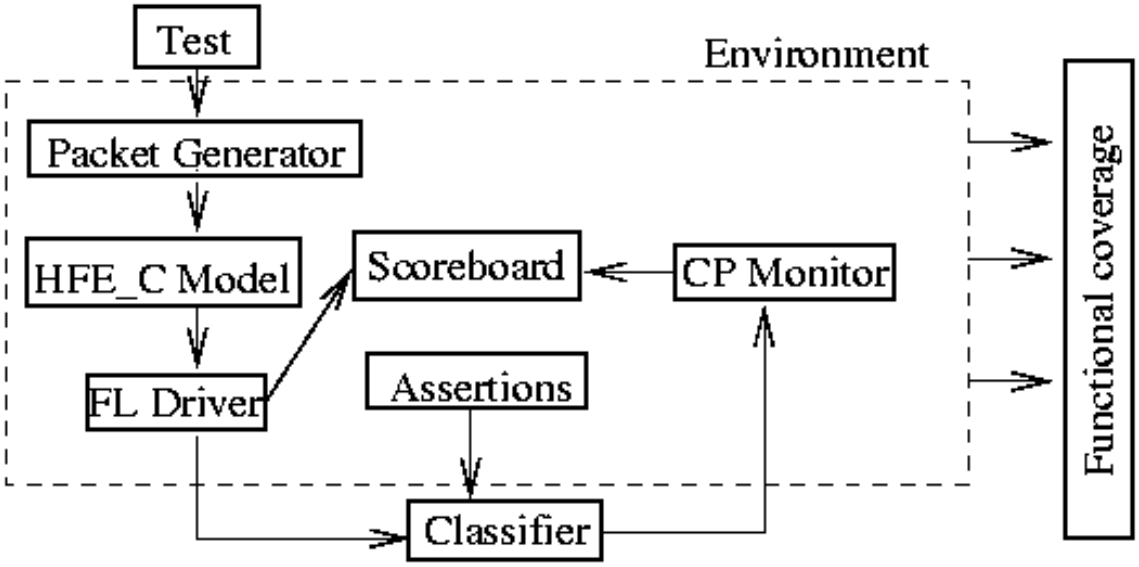
Na obrázku v příloze B je třídní diagram zobrazující třídy síťových protokolů. Předkem všech těchto tříd je abstraktní vrstva Layer, která sdružuje všechny společné atributy a metody. Blíže se budeme těmito třídami zabývat v implementační sekci.

6.1.4 Infrastruktura pro nestandardní rozhraní

Classifier a Ptrn_match obsahují nestandardní rozhraní. Jedná se o rozhraní mezi jednotkou Classifier a Ptrn_match a výstup jednotky Ptrn_match. Obě rozhraní jsou vcelku jednoduchá. První z nich se skládá ze signálu o šířce 5000 bitů, které udává výsledek klasifikace a ze signálů určujících platnost klasifikace a jejího přijetí. Je třeba vytvořit pro protokol

Protokol	Popis
Ethernet II	Protokol Erhernetu verze II.
Ethernet II s IEEE 802.1Q	Protokol Erhernetu verze II. s podporou VLAN
IPv4	Internet protokol verze 4
IPv6	Internet protokol verze 6
ICMP	ICMP protokol pro IPv4
ICMPv6	ICMP protokol pro IPv6
TCP	Protokol transporní vrstvy spojově orientovný
UDP	Protokol transportní vrstvy nespojově orientovaný
RAW	Surová náhodná data
PatternRAW	Surová náhodná data s možností vložení vzorů

Tabulka 6.1: Podporované protokoly



Obrázek 6.3: Schéma pro verifikaci komponenty Classifier.

tohoto rozhraní transakci, driver a monitor. Druhé rozhraní je ještě jednodušší, skládá ze jednobitového signálu určujícího, zda paket splnil některé pravidlo a ze signálu potvrzujícího jeho platnost. Pro protokol toho výstupního rozhraní je třeba vytvořit transakci a monitor.

6.1.5 Scoreboardy

Pro scoreboardy je třeba vytvořit modely chování jednotlivých testovaných komponent. Pro komponentu HFE_C to znamená, že je třeba provést extrakci hlaviček z příchozí transakce a vytvořit dvě výstupní transakce typu FrameLink. Jednu pro rozhraní unifikovaných hlaviček a druhé pro rozhraní STOR. Pro komponentu Classifier to znamená zpracovat vstupní transakci s UH hlavičkami a zjistit jestli došlo ke splnění pravidel. Je nutné zjistit výsledky pro všechna pravidla, poněvadž se dále zasílá transakce s touto informací. Model jednotky Ptrn_match musí vyhledat v paketu zadané vzory a za použití informací z klasifikátoru rozhodnout zda paket spnil pravidlo, nebo ho nesplnil. Tato informace je pak odeslána k porovnání s výstupem komponenty.

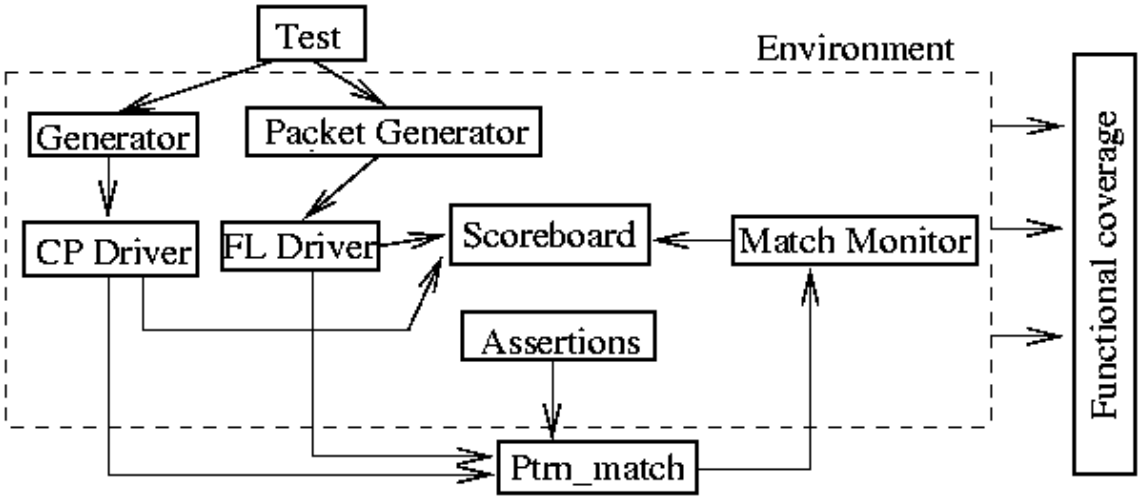
6.2 Implementace

Následující text popisuje jednotlivé důležité třídy systému.

6.2.1 PacketGenerator

Popis

Paketový generátor dědí od třídy Generator, která představuje obecný generátor. K funkcionalitě tohoto obecného generátoru přidává vlastnosti, které byly popsány výše v návrhu. Diagram tříd můžete nalézt na obrázku v příloze 6.7. Třída je implementována v souboru packetGenerator.sv.



Obrázek 6.4: Schéma pro verifikaci komponenty Ptrn_Match.

Zděděné atributy

tTransMbx transMbx; Mailbox pro výstup vygenerovaných transakcí.

int unsigned stop_after_n_insts = 0; Udává po kolika vygenerovaných transakcích se generátor zastaví. Nula znamená, že se nezastaví sám nikdy.

Transaction blueprint; Vzor transakce, který se bude randomizovat.

bit enabled; Nastavuje zda je generátor zapojen.

Atributy

LayerP [] protocols; Dynamické pole obsahující vložené síťové protokoly spolu s jejich pravděpodobností.

int protocolCount; Počet aktuálně vložených protokolů.

Zděděné metody

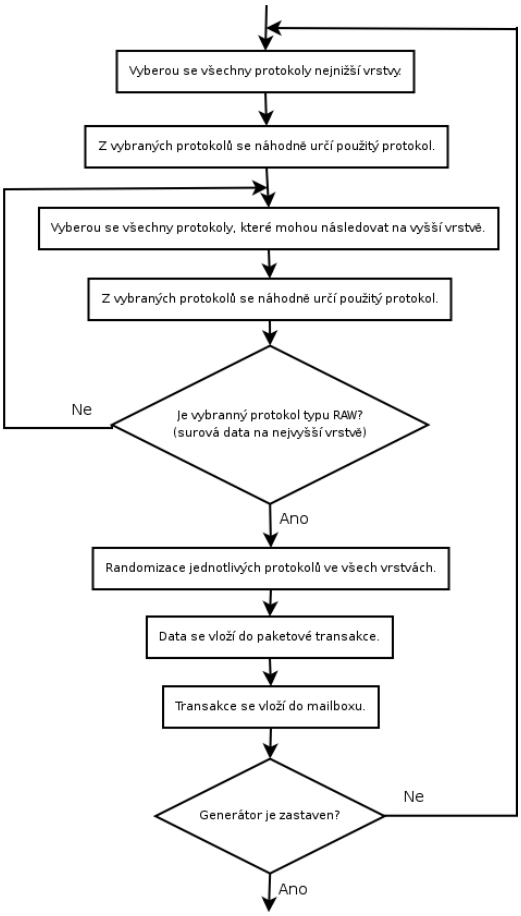
task setEnabled(int unsigned nInst=32'hFFFFFFFF); Spustí generátor pro vygenerování nInst transakcí.

task setDisabled(); Zastaví generátor.

Metody

functon new(); Konstruktor třídy. Vytvoří dynamické pole o velikosti 5 prvků a nastaví počet protokolů na nulu.

function bit addProtocol(Layer protocol, int probability); Vloží do generátoru protokol společně s jeho pravděpodobností. V případě potřeby zvětší prostor pro uložení těchto informací a zvětší čítač počtu vložených protokolů o jedničku.



Obrázek 6.5: Vývojový diagram činnosti spušťeného paketového generátoru.

virtual task run(); Spouští generátor, vytváří v cyklu nová vlákna generátoru, která generují transakce. Běží do té doby dokud není zastaven, nebo není překročen zadaný počet generovaných transakcí.

6.2.2 Třída Layer

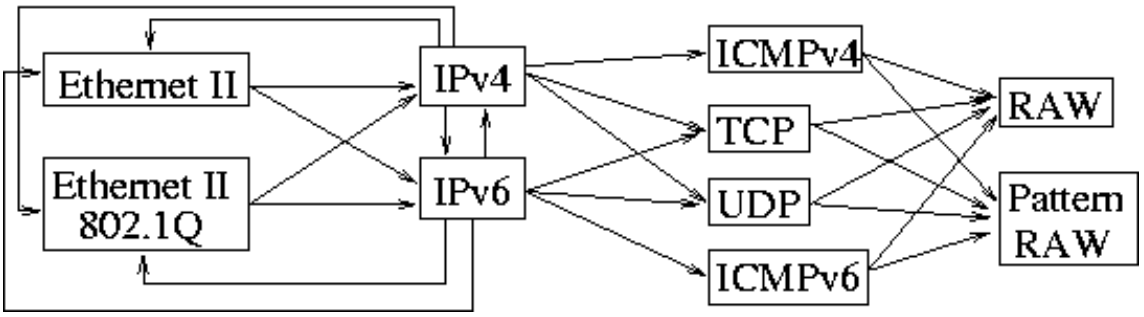
Popis

Tato třída představuje bázovou abstraktní třidu pro všechny protokoly, čímž umožňuje to, aby se navenek chovaly stejně. Funkce jednotlivých virtualních metod a společných atributů bude probrána zde a nebude se opakovat u jednotlivých dceřiných tříd. Třída je implementována v souboru layer.sv. Objekty tříd protokolů spolu tvoří seznam, takže stačí některou operaci jako je randomizace, získání dat a jejich velikosti spustit na nejnižší vrstvě a tato vrstva si zjistí informace z vyšší vrstev sama.

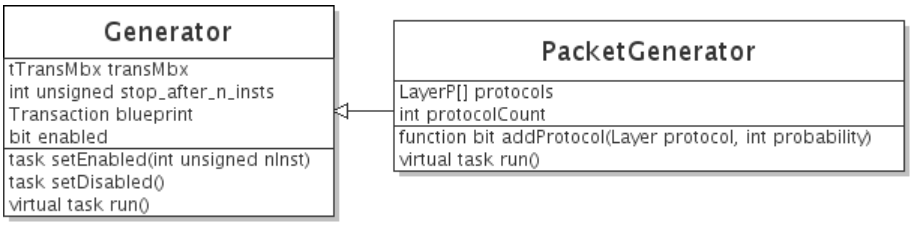
Atributy

string type; Typ protokolu (např. Ethernet, IPv4, RAW, ...).

string subtype; Podtyp protokolu.



Obrázek 6.6: Vazby protokolů.



Obrázek 6.7: Diagram tříd paketového generátoru.

string name; Jméno třídy.

int layer; Číslo vrstvy. Používá se pro výběr nejnižší vrstvy.

Layer next; Handle objektu následující vrstvy.

Layer previous; Handle objektu předcházející vrstvy.

int errorProbability; Pravděpodobnost generování chyby v protokolu.

int minMTU; Minimální velikost dat požadovaná pro tuto vrstvu a vrstvy vyšší.

int maxMTU; Maximální velikost dat požadovaná pro tuto vrstvu a vrstvy vyšší.

Metody

virtual function byte unsigned[] getHeader(); Vrátí pole bezeznaménkových bytů obsahující hlavičku.

virtual function byte unsigned[] getFooter(); Vrátí pole bezeznaménkových bytů obsahující patičku.

virtual function byte unsigned[] getAttributeByName(string name); Vrátí pole bezeznaménkových bytů obsahující obsah položky protokolu podle jména kompatibilního s názvem v HFE_C.

virtual function byte unsigned[] getData(); Získá pomocí funkcí getHeader a getFooter data protokolu a mezi ně vloží data získaná z vyšší vrstvy.

virtual function Layer copy(); Zkopíruje třídu.

virtual function bit checkType(string type, string subtype ,string name); Tato funkce slouží pro kontrolu, zda může být vyšší vrstva použita spolu s danou vrstvou.

virtual function void display(); Zobrazí obsah dané třídy.

virtual function int getLength(bit split); Vrátí velikost dat tohoto protokolu plus velikost vyšších vrstev. Pokud je split nastaven na 1 musí vrstva typu RAW vrátit nulovou velikost.

6.2.3 Ethernet_II

Popis

Tato třída dědí od třídy Layer a implementuje síťový protokol Ethernet II. Popis protokolu je možné najít v [20]. Tato třída má typ ETHERNET, subtyp II a jméno Ethernet II. Umožňuje nastavit rozsah náhodného generování zdrojové a cílové MAC adresy. Defaultně je tento rozsah od minimální možné hodnoty po maximální možnou hodnotu. Implementace se nachází v souboru ethernet_II.sv.

Atributy

bit [47:0] minSrcMAC; Dolní hranice pro generování zdrojové MAC adresy.

bit [47:0] maxSrcMAC; Horní hranice pro generování zdrojové MAC adresy.

bit [47:0] minDstMAC; Dolní hranice pro generování cílové MAC adresy.

bit [47:0] maxDstMAC; Horní hranice pro generování cílové MAC adresy.

rand bit [47:0] destinationMAC; Cílová MAC adresa. Náhodně generované číslo leží v zadaném intervalu.

rand bit [47:0] sourceMAC; Zdrojová MAC adresa. Náhodně generované číslo leží v zadaném intervalu.

bit [15:0] type; Identifikace protokolu vyšší vrstvy. Je nastaveno podle aktuálního protokolu vyšší při randomizaci. Třídou podporované protokoly jsou IPv4 a IPv6. Výčet podporovaných je možné nalézt v [9]

rand bit [31:0] CRC; Náhodně generovaný kontrolní součet.

6.2.4 Ethernet_II_dot1q

Popis

Tato třída dědí od třídy Ethernet_II a přidává k Ethernetu II podporu VLAN podle standardu IEEE 802.1Q [20]. Tato třída má typ ETHERNET, subtyp II 802.1Q a jméno Ethernet II 802.1Q. Pole hlavičky type je nastaveno na hodnotu 0x8100, která indikuje, že je připojena 16-bitová informace o VLAN [9]. Implementace se nachází v souboru ethernet_II_dot1q.sv.

Atributy

rand bit [15:0] TCI Toto pole sdružuje informace o VLAN. Bity [11:0] obsahují identifikátor VLAN, jehož hodnota se při randomizaci nastaví na hodnotu ležící v intervalu (0,4095). Hodnoty identifikátoru VLAN 0 a 4095 jsou rezervovanými hodnotami podle [20].

6.2.5 IPv4

Popis

Tato třída dědí od třídy Layer a implementuje síťový protokol IP verze 4. Popis tohoto protokolu je možné najít v RFC 791 [13] nebo souhrně v [23]. Tato třída má typ IP, subtyp 4 a jméno IPv4. Umožňuje nastavit rozsah náhodného generování zdrojové a cílové IP adresy. Defaultně je tento rozsah od minimální možné hodnoty po maximální možnou hodnotu. Nejsou podporovány volitelné hlavičky. Implementace se nachází v souboru ipv4.sv.

Atributy

bit [31:0] minSrcIP; Dolní hranice pro generování zdrojové IP adresy.

bit [31:0] maxSrcIP; Horní hranice pro generování zdrojové IP adresy.

bit [31:0] minDstIP; Dolní hranice pro generování cílové IP adresy.

bit [31:0] maxDstIP; Horní hranice pro generování cílové IP adresy.

rand bit [3:0] version; Verze protokolu IP. Vždy 4 pro IP verze 4.

rand bit [3:0] headerLength; Délka hlavičky ve 32-bitových slovech. Bez volitelných hlaviček vždy 5.

rand bit [7:0] typeOfService; V původní specifikaci určeno pro typ služby, nyní se používá pro jiné účely [23].

bit [15:0] totalLength; Celková délka IP paketu. Do délky se počítá hlavička a délka dat vyšších vrstev. Minimální velikost je 20 bytů.

rand bit [15:0] identification; Identifikuje fragmenty původního paketu.

rand bit [2:0] flags; Ukazuje zda je povolena fragmentace a jestli je paket fragmentován. Nejvyšší bit musí být nulový [23].

rand bit [12:0] fragmentOffset; Ofset fragmentu vztažený k původnímu paketu. Udává se v 8 bytových blocích.

rand bit [7:0] timeToLive; Doba života paketu.

bit [7:0] protocol; Protocol vyšší vrstvy. Podporovány třídou jsou Ethernet, IPv4, IPv6, ICMPv4, UDP, TCP. Přehled podporovaných protokolů je možné najít v [8].

rand bit [15:0] headerChecksum; Náhodně generovaný kontrolní součet.

rand bit [31:0] sourceAddress; Zdrojová IP adresa. Náhodně generované číslo leží v zadaném intervalu.

rand bit [31:0] destinationAddress; Cílová IP adresa. Náhodně generované číslo leží v zadaném intervalu.

6.2.6 IPv6

Popis

Tato třída dědí od třídy Layer a implementuje síťový protokol IP verze 6. Popis tohoto protokolu je možné najít v RFC 2460 [3] nebo souhrnně v [25]. Tato třída má typ IP, subtyp 6 a jméno IPv6. Umožňuje nastavit rozsah náhodného generování zdrojové a cílové IP adresy. Defaultně je tento rozsah od minimální možné hodnoty po maximální možnou hodnotu. Nejsou podporovány volitelné hlavičky. Implementace se nachází v souboru ipv6.sv.

Atributy

bit [127:0] minSrcIP; Dolní hranice pro generování zdrojové IP adresy.

bit [127:0] maxSrcIP; Horní hranice pro generování zdrojové IP adresy.

bit [127:0] minDstIP; Dolní hranice pro generování cílové IP adresy.

bit [127:0] maxDstIP; Horní hranice pro generování cílové IP adresy.

rand bit [3:0] version; Verze protokolu IP. Vždy 6 pro IP verze 6.

rand bit [7:0] trafficClass; Priorita paketu.

rand bit [19:0] flowLabel; Řízení kvality služeb. Nyní nevyužíváno viz [25].

bit [15:0] payloadLength; Velikost dat vyšší vrstvy.

rand bit [7:0] nextHeader; Následující protokol vyšší vrstvy. Hodnoty jsou kompatibilní s IPv4. Podporovány třídou jsou Ethernet, IPv4, IPv6, ICMPv6, UDP, TCP. Přehled podporovaných protokolů je možné najít v [8].

rand bit [7:0] hopLimit; Doba života paketu.

rand bit [127:0] sourceAddress; Zdrojová IP adresa. Náhodně generované číslo leží v zadaném intervalu.

rand bit [127:0] destinationAddress; Cílová IP adresa. Náhodně generované číslo leží v zadaném intervalu.

6.2.7 ICMP

Popis

Tato třída dědí od třídy Layer a implementuje síťový kontrolní protokol ICMP pro protokol IP verze 4. Popis tohoto protokolu je možné najít v RFC 792 [12] nebo souhrnně v [22]. Tato třída má typ ICMP, subtyp 4 a jméno ICMPv4. Jediným protokolem vyšší vrstvy, který může následovat je protokol typu RAW, tj. surová data. Nejsou podporovány formátovaná data některých správ ICMP. Implementace se nachází v souboru icmp.sv.

Atributy

rand bit [7:0] type; Typ zprávy protokolu ICMP. Randomizace respektuje existující typy zpráv podle tabulky v [6].

rand bit [7:0] code; Kód bližší specifikace typu zprávy. Randomizace respektuje existující upřesňující kódy podle tabulky v [6].

rand bit [15:0] checksum; Náhodně generovaný kontrolní součet.

rand bit [15:0] id; Identifikační číslo.

rand bit [15:0] sequence; Sekvenční číslo.

6.2.8 ICMPv6

Popis

Tato třída dědí od třídy Layer a implementuje síťový kontrolní protokol ICMP pro protokol IP verze 6. Popis tohoto protokolu je možné najít v RFC [2] nebo souhrnně [21]. Tato třída má typ ICMP, subtyp 6 a jméno ICMPv6. Jediným protokolem vyšší vrstvy, který může následovat je protokol typu RAW, tj. surová data. Nejsou podporovány formátovaná data některých správ ICMP. Implementace se nachází v souboru icmp.sv.

Atributy

rand bit [7:0] type; Typ zprávy protokolu ICMP. Randomizace respektuje existující typy zpráv podle tabulky v [7].

rand bit [7:0] code; Kód bližší specifikace typu zprávy. Randomizace respektuje existující upřesňující kódy podle tabulky v [7].

rand bit [15:0] checksum; Náhodně generovaný kontrolní součet.

6.2.9 TCP

Popis

Tato třída dědí od třídy Layer a implementuje síťový transportní protokol TCP. Popis tohoto protokolu je možné najít v RFC 793 [14] nebo souhrnně v [27]. Tato třída má typ TCP a jméno TCP. Umožňuje nastavit rozsah náhodného generování zdrojového a cílového portu. Defaultně je tento rozsah od minimální možné hodnoty po maximální možnou hodnotu. Jediným protokolem vyšší vrstvy, který může následovat je protokol typu RAW, tj. surová data. Nejsou podporovány volitelné hlavičky. Implementace se nachází v souboru udp.sv.

Atributy

bit [15:0] minSrcPort; Dolní hranice pro generování zdrojového portu.

bit [15:0] maxSrcPort; Horní hranice pro generování zdrojového portu.

bit [15:0] minDstPort; Dolní hranice pro generování cílového portu.

bit [15:0] maxDstPort; Horní hranice pro generování cílového portu.

rand bit [15:0] sourcePort; Zdrojový port. Náhodně generované číslo leží v zadaném intervalu.

rand bit [15:0] destinationPort; Cílový port. Náhodně generované číslo leží v zadaném intervalu.

rand bit [31:0] sequenceNumber; Sekvenční číslo.

rand bit [31:0] acknowledgmentNumber; Potvrzení přijetí dat s udaným sekvenčním číslem.

rand bit [3:0] dataOffset; Velikost hlavičky ve 32-bitových slovech. Bez rozšířených hlaviček 5, maximálně 15.

rand bit [3:0] reserved; Rezervováno.

rand bit [7:0] flags; Kontrolní bity. Jejich význam viz. [27].

rand bit [15:0] windowSize; Velikost příjímacího okna.

rand bit [15:0] checksum; Náhodně generovaný kontrolní součet.

rand bit [15:0] urgentPointer; Ofset posledního urgentního bytu.

6.2.10 UDP

Popis

Tato třída dědí od třídy Layer a implementuje síťový transportní protokol UDP. Popis tohoto protokolu je možné najít v RFC 768 [15] nebo souhrnně v [28]. Tato třída má typ UDP a jméno UDP. Umožňuje nastavit rozsah náhodného generování zdrojového a cílového portu. Defaultně je tento rozsah od minimální možné hodnoty po maximální možnou hodnotu. Jediným protokolem vyšší vrstvy, který může následovat je protokol typu RAW, tj. surová data. Implementace se nachází v souboru udp.sv.

Atributy

bit [15:0] minSrcPort; Dolní hranice pro generování zdrojového portu.

bit [15:0] maxSrcPort; Horní hranice pro generování zdrojového portu.

bit [15:0] minDstPort; Dolní hranice pro generování cílového portu.

bit [15:0] maxDstPort; Horní hranice pro generování cílového portu.

rand bit [15:0] sourcePort; Zdrojový port. Náhodně generované číslo leží v zadaném intervalu.

rand bit [15:0] destinationPort; Cílový port. Náhodně generované číslo leží v zadaném intervalu.

bit [15:0] length; Délka celého datagramu (hlavička a data vyšší vrstvy).

rand bit [15:0] checksum; Náhodně generovaný kontrolní součet.

6.2.11 RAW

Popis

Tato třída dědí od třídy Layer a implementuje surová náhodná data. Velikost vygenerovaných dat je náhodně určena v rámci daných mezí. Tato třída má typ RAW a jméno RAW. Implementace se nachází v souboru raw.sv.

Atributy

byte unsigned[] data; Náhodná data o náhodně stanovené délce v rámci nastavených mezí.

6.2.12 RAWPattern

Popis

Tato třída dědí od třídy RAW a implementuje surová náhodná data, do kterých jsou přidány na náhodná místa zadané vzory. Každý vzor má nastavenu pravděpodobnost s jakou se použije. Pravděpodobnost se počítá stejně jako u Paketového generátoru. Tato třída vznikla za účelem verifikace jednotky Patter_match. Velikost vygenerovaných dat je náhodně určena v rámci daných mezí. Tato třída má typ RAW, subtyp Pattern a jméno RAW with patterns. Implementace se nachází v souboru raw_pattern.sv.

Atributy

TPattern[] patterns; Vložené vzory spolu s jejich pravděpodobností.

int patternCount; Počet vložených vzorů.

int probabilitySum; Součet pravděpodobností.

Metody

function void addPattern(string pattern, int probability); Vloží vzor se zadanou pravděpodobností.

6.2.13 Další třídy

Pro rozhraní mezi jednotkami Classifier a Ptrn_match je implementována třída CPTransaction pro transakci, CPDriver pro driver a CPMonitor pro monitor. Pro výstup jednotky Ptrn_match je implementován driver ve třídě MatchDriver a transakce ve třídě MatchTransaction.

HFE_C model zmiňovaný na obrázku 6.3 vychází z modelu HFE_C pro scoreboard používaný při verifikaci HFE_C. Je však upraven pro samostatné použití a protože je mezi generátorem a driverem, tak s okolím komunikuje přes vstupní a výstupní mailbox.

6.2.14 Stav implementace systému

Paket generátor je naimplementován a při verifikaci HFE_C odzkoušen. Stejně jsou implementovány i třídy rozhranní jednotek Classifier a Ptrn_match. Verifikace jednotky Classifier je naimplementována, úspěšné verifikaci zatím brání neopravená chyba ve FrameLinkových verifikačních komponentách. Implementace verifikace jednotky Ptrn_match je plánována po ukončení verifikace jednotky Classifier.

Kapitola 7

Závěr

7.1 Možná rozšíření

7.1.1 Podpora dalších protokolů

Jedním z možných rozšíření je přidání podpory pro další protokoly jako je ARP, IGMP apod. Přidáním dalších protokolů by bylo možné provádět důkladnější testy funkcionality. Dále by bylo možné přidat podporu pro volitelné hlavičky protokolů IPv4, IPv6 a TCP, připadně počítání CRC, které je potřeba pro testy komponenty IBUF.

7.1.2 Automatické generování modelu jednotky HFE_C

V systému IDS se komponenta HFE_C používá v jedné konfiguraci, takže je její model provádějící predikci výstupů testované komponenty naprogramován napevno. Tato komponenta se však používá i v dalších projektch jako NIFIC (viz [33]) nebo FlexibleFlowmon (viz [35]). V projektu FlexibleFlomon je navíc používána v různých konfiguracích podle aktuálně požadované činnosti (viz [35]), tudíž by bylo vhodné rozšířit systém o podporu automatického generování modelu pro verifikaci na základě XML popisu, který je používán v projektu FlexibleFlomon pro popis konfigurace HFE_C.

7.1.3 Automatické zpracování pravidel systému Snort

Část kódu komponent Classifier [29] a Ptrn_match je generována na základě pravidel systému Snort, které udávají funkcionalitu těchto jednotek. Proto by bylo vhodné vytvořit systém pro automatické generování modelů pro tyto jednotky, aby bylo možné každý vytvořený design také ihned zverifikovat.

7.2 Zhodnocení

Výše popsaný systém umožňuje zlepšit testováví síťových komponent. Je dostatečně modulární, aby ho bylo možné použít s drobnými úpravami i pro testování jednotek s jiným rozhraním než je FrameLink(Například (X)GMII pro komponentu IBUF). Přínos tohoto systému s paketovým generátorem spočívá v tom, že umožňuje jednoduše testovat složité komponenty zpracovávající síťový provoz. Současné řešení spočívá v simulaci komponenty s několika pakety (řádově jednotky až desítky) a hlavní testování se dělá až v hardware, jelikož testování komponenty tradičním způsobem (řízené testování) s velkým počtem různých

paketů je náročné na lidské zdroje. Automatizace takového testu umožní návrháři (po vytvoření modelu komponenty pro predikci výstupů reálné testované jednotky, sestavení testbenche a návrhu testů) ušetřit práci, kterou by při tradičním způsobu testování musel věnovat na manuální přípravu testů a analýzu výsledků.

Zvolená metodologie prokázala svoji užitečnost, když se podařilo odhalit chybu v jednotce HFE_C na výstupním rozhraní STOR. Tato chyba by jinak pravděpodobně unikla pozornosti, poněvadž se jedná o to, že se nesprávně rozděluje paket na hlavičky a užitečná data. Odhalení této chyby běžným přístupem by znamenalo ručně projít celý paket, zpracovat hlavičky protokolů a určit tak očekávaný výsledek, což by zabralo značné množství času. Při použití zvolené verifikační metodologie byla chyba odhalena automaticky.

Po dokončení verifikace použitých obecných komponent bude (podle zvolené metodologie) možné přistoupit k verifikaci designu systému pro detekci nežádoucího provozu jako celku.

Literatura

[1] Bergeron, J. aj. *Verification Methodology Manual for SystemVerilog*. Springer Science+Business Media, LLC, 2005. ISBN 0-387-25538-9.

[2] Conta, A. aj. Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification. `http://tools.ietf.org/html/rfc4443` (12.5.2008). RFC 4443.

[3] Deering, S. aj. Internet protocol, version 6 (ipv6) specification. `http://tools.ietf.org/html/rfc2460` (12.5.2008). RFC 2460.

[4] Kobierský, P. aj. Systemverilog verification of vhdl design. Technical Report 35, CESNET z. s. p. o., 2007. (12.5.2008).

[5] Spear, Ch. *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Springer Science+Business Media, LLC, 2006. ISBN 0-387-27036-1.

[6] IANA. Icmp type numbers. `http://www.iana.org/assignments/icmp-parameters`. (12.5.2008).

[7] IANA. Internet control message protocol version 6 (icmpv6) type numbers. `http://www.iana.org/assignments/icmpv6-parameters`. (12.5.2008).

[8] IANA. Protocol numbers. `http://www.iana.org/assignments/protocol-numbers`. (12.5.2008).

[9] IEEE. Ieee list of ethertype values. `http://standards.ieee.org/regauth/ethertype/eth.txt`. (12.5.2008).

[10] Accellera Organization, Inc. Systemverilog 3.1a language reference manual - accellera's extensions to verilog. `http://www.eda.org/sv/SystemVerilog_3.1a.pdf`, 2004. (12.5.2008).

[11] Bergeron, J. *Writing Testbenches Using SystemVerilog*. Springer Science+Business Media, LLC, 2006. ISBN 0-387-29221-7.

[12] Postel, J. Internet control message protocol. `http://tools.ietf.org/html/rfc792` (12.5.2008). RFC 792.

[13] Postel, J. Internet protocol. `http://tools.ietf.org/html/rfc791` (12.5.2008). RFC 791.

[14] Postel, J. Transmission control protocol. http://tools.ietf.org/html/rfc793 (12.5.2008). RFC 793.

[15] Postel, J. User datagram protocol. http://tools.ietf.org/html/rfc768 (12.5.2008). RFC 768.

[16] P. Kobierský. *Návrh architektury IDS systému pro technologii FPGA*. FIT VUT Brno, 2006. Bakalářská práce.

[17] Hank, A. Kobierský, J., Kořenek, J. Traffic scanner. Technical Report 33, CESNET z. s. p. o., 2006. (12.5.2008).

[18] Snort. Snort news. http://www.snort.org/. (12.5.2008).

[19] IEEE std. 1800 2005. *IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language*. The Institute of Electrical and Electronics Engineers, Inc., 2005. ISBN 0-7381-4811-3.

[20] Wikipedia. Ethernet — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Ethernet&oldid=211689965, 2008. (12.5.2008).

[21] Wikipedia. Icmpv6 — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=ICMPv6&oldid=209178073, 2008. (12.5.2008).

[22] Wikipedia. Internet control message protocol — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Internet_Control_Message_Protocol&oldid=211834030, 2008. (12.5.2008).

[23] Wikipedia. Ipv4 — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=IPv4&oldid=210985004, 2008. (12.5.2008).

[24] Wikipedia. Ipv4 mapped address — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=IPv4_mapped_address&oldid=208840792, 2008. (12.5.2008).

[25] Wikipedia. Ipv6 — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=IPv6&oldid=211833937, 2008. (12.5.2008).

[26] Wikipedia. Systemverilog — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=SystemVerilog&oldid=203542504, 2008. (12.5.2008).

[27] Wikipedia. Transmission control protocol — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&oldid=211588497, 2008. (12.5.2008).

[28] Wikipedia. User datagram protocol — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=User_Datagram_Protocol&oldid=211748397, 2008. (12.5.2008).

[29] CESNET z. s. p. o. Classifier for ids. http://www.liberouter.org/vhdl_design/generated/HEAD_classifier_info.php. (12.5.2008).

[30] CESNET z. s. p. o. Framelink. https://www.liberouter.org/wiki/index.php/FrameLink (12.5.2008). interní materiál.

[31] CESNET z. s. p. o. Header field extractor in handel-c. http://www.liberouter.org/vhdl_design/generated/HEAD_hfe_c_info.php. (12.5.2008).

[32] CESNET z. s. p. o. Netcope. http://www.liberouter.org/netcope/index.php. (12.5.2008).

[33] CESNET z. s. p. o. Nific. http://www.liberouter.org/nific.php. (12.5.2008).

[34] CESNET z. s. p. o. Verifikace fl komponent. https://www.liberouter.org/trac/firmware/wiki/verifikace-fl-komponent (12.5.2008). interní materiál.

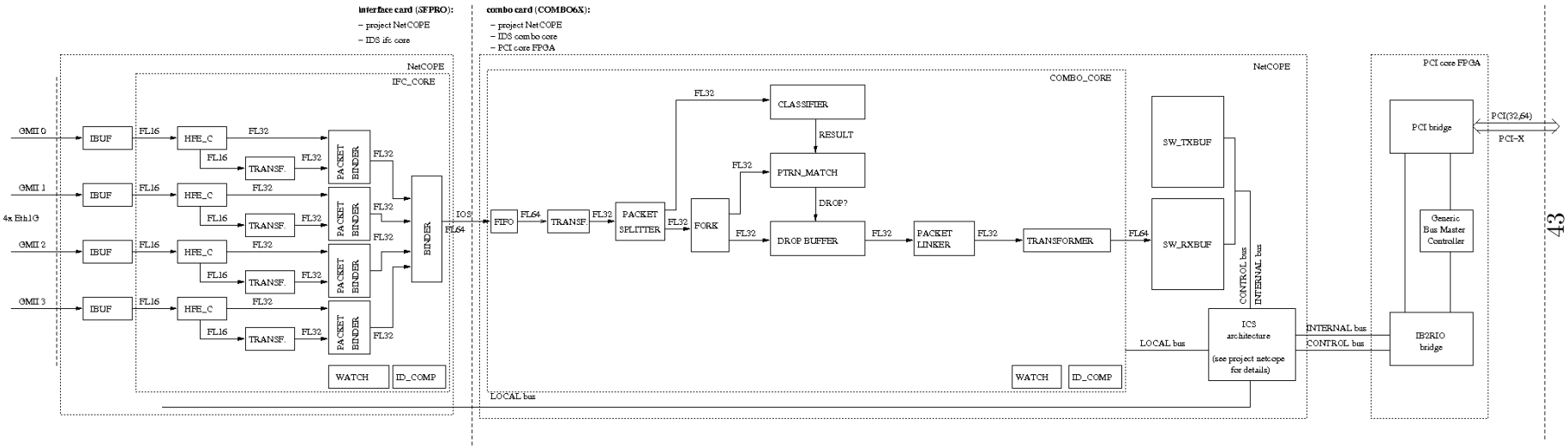
[35] Čeleda, P. Žádnik, M., Špringl, P. Flexible flowmon. Technical Report 36, CESNET z. s. p. o., 2007. (12.5.2008).

Seznam příloh:

- A Detailní schéma systému pro detekci nežádoucího provozu.
- B Diagram tříd síťových protokolů.
- C Seznam použitých zkratek.
- D Návod na spuštění.
- E CD-ROM

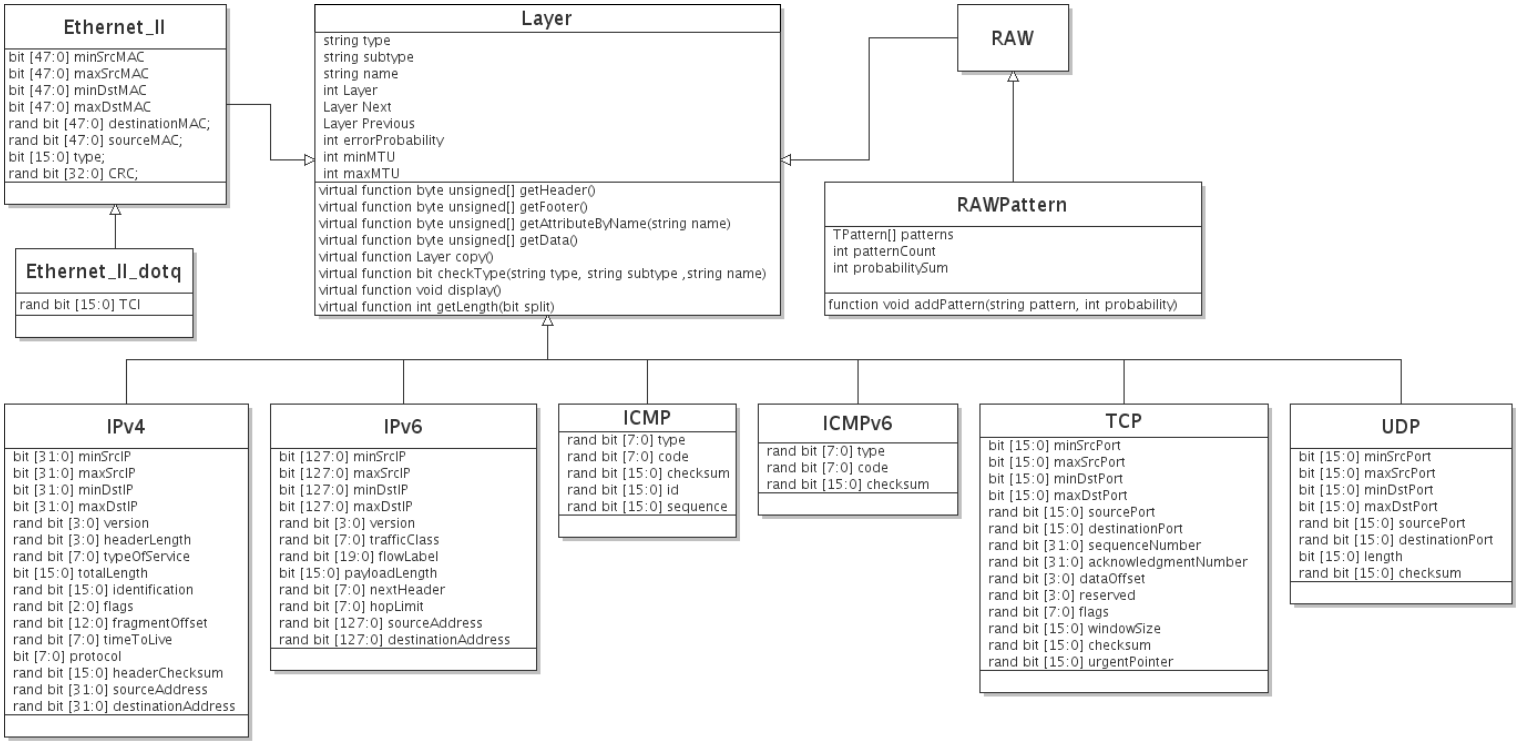
Dodatek A

Detailní schéma systému pro detekci nežádoucího provozu


interface card (SFPRO):
– project NetCOPE
– IDS ifc core
combo card (COMBO6X):
– project NetCOPE
– IDS combo core
– PCI core FPGA
NetCOPE
IFC_CORE
GMII 0
GMII 1
4x Eth1G
GMII 2
GMII 3
IBUF
HFE_C
TRANSF.
PACKET BINDER
BINDER
WATCH
ID_COMP
FL16
FL32
FL64
IOS
LOCAL bus
COMBO_CORE
FIFO
TRANSF.
PACKET SPLITTER
FORK
CLASSIFIER
RESULT
PTRN_MATCH
DROP?
DROP BUFFER
PACKET LINKER
TRANSFORMER
SW_TXBUF
SW_RXBUF
CONTROL bus
INTERNAL bus
ICS architecture
(see project netcope for details)
PCI core FPGA
PCI bridge
PCI(32,64)
PCI-X
Generic Bus Master Controller
IB2RIO bridge

Dodatek B

Diagram tříd síťových protokolů


Ethernet_II
bit [47:0] minSrcMAC
bit [47:0] maxSrcMAC
bit [47:0] minDstMAC
bit [47:0] maxDstMAC
rand bit [47:0] destinationMAC;
rand bit [47:0] sourceMAC;
bit [15:0] type;
rand bit [32:0] CRC;
Ethernet_II_dotq
rand bit [15:0] TCI
Layer
string type
string subtype
string name
int Layer
Layer Next
Layer Previous
int errorProbability
int minMTU
int maxMTU
virtual function byte unsigned[] getHeader()
virtual function byte unsigned[] getFooter()
virtual function byte unsigned[] getAttributeByName(string name)
virtual function byte unsigned[] getData()
virtual function Layer copy()
virtual function bit checkType(string type, string subtype ,string name)
virtual function void display()
virtual function int getLength(bit split)
RAW
RAWPattern
TPattern[] patterns
int patternCount
int probabilitySum
function void addPattern(string pattern, int probability)
IPv4
bit [31:0] minSrcIP
bit [31:0] maxSrcIP
bit [31:0] minDstIP
bit [31:0] maxDstIP
rand bit [3:0] version
rand bit [3:0] headerLength
rand bit [7:0] typeOfService
bit [15:0] totalLength
rand bit [15:0] identification
rand bit [2:0] flags
rand bit [12:0] fragmentOffset
rand bit [7:0] timeToLive
bit [7:0] protocol
rand bit [15:0] headerChecksum
rand bit [31:0] sourceAddress
rand bit [31:0] destinationAddress
IPv6
bit [127:0] minSrcIP
bit [127:0] maxSrcIP
bit [127:0] minDstIP
bit [127:0] maxDstIP
rand bit [3:0] version
rand bit [7:0] trafficClass
rand bit [19:0] flowLabel
bit [15:0] payloadLength
rand bit [7:0] nextHeader
rand bit [7:0] hopLimit
rand bit [127:0] sourceAddress
rand bit [127:0] destinationAddress
ICMP
rand bit [7:0] type
rand bit [7:0] code
rand bit [15:0] checksum
rand bit [15:0] id
rand bit [15:0] sequence
ICMPv6
rand bit [7:0] type
rand bit [7:0] code
rand bit [15:0] checksum
TCP
bit [15:0] minSrcPort
bit [15:0] maxSrcPort
bit [15:0] minDstPort
bit [15:0] maxDstPort
rand bit [15:0] sourcePort
rand bit [15:0] destinationPort
rand bit [31:0] sequenceNumber
rand bit [31:0] acknowledgmentNumber
rand bit [3:0] dataOffset
rand bit [3:0] reserved
rand bit [7:0] flags
rand bit [15:0] windowSize
rand bit [15:0] checksum
rand bit [15:0] urgentPointer
UDP
bit [15:0] minSrcPort
bit [15:0] maxSrcPort
bit [15:0] minDstPort
bit [15:0] maxDstPort
rand bit [15:0] sourcePort
rand bit [15:0] destinationPort
bit [15:0] length
rand bit [15:0] checksum

Dodatek C

Seznam použitých zkratek

IDS: Intrusion detection system.

DUT: Design under test.

CRT: Constrained-random test.

CAM: Content-addressable memory.

IPv4: Internet protocol version 4.

IPv6: Internet protocol version 4.

ICMP, ICMPv4: Internet control message protocol for Internet protocol version 4.

ICMPv6: Internet control message protocol for Internet protocol version 6.

TCP: Transmission control protocol.

UDP: User datagram protocol.

IGMP: Internet group management protocol.

ARP: Address resolution protocol.

RARP: Reverse address resolution protocol.

Dodatek D

Návod na spuštění

Pro spuštění verifikace je třeba mít nainstalovaný nástroj Modelsim verze nejlépe 6.3f, na operačním systému Linux, nebo v prostředí Cygwin ve Windows.

Verifikace jednotlivých komponent se nachází v následujících adresářích:

HFE_C: `ids/trunk/firmware/comp/hfe_c/ver/`

Classifier: `ids/trunk/firmware/comp/ipv6_classifier/ver/`

Pro spuštění verifikace je třeba zadat příkaz:

```
vsim -do top_level.fdo
```

Simulaci jednotky CLASSIFIER s podporou IPv6 je možné spustit příkazem:

```
vsim -do top_level.fdo
```

Ilustrativní je verifikace HFE_C, kde je ukázána výhoda CRT při verifikaci. Jediným vygenerovaným paketem s protokolem transportní vrstvy UDP je odhalena výše uvedená chyba.Chybu zjitíme z výpisu v okně Modelsimu Transcript. Ve výpise se nachází výpis obsahu transakční tabulky. Nejdříve je vypsána transakce přijatá od monitoru, jejíž predikovaná hodnota se nachází v tabulce. Tato transakce je označena jako neodpovídající žádné transakci v tabulce. A z rozdílu délky jednotlivých paketů rámce FrameLinku zjistíme chybu.

```
# Unknown transaction received from monitor Monitor1
# Packet no:           0, Packet size:           4, Data: 00000000
# Packet no:           1, Packet size:          34, Data: b90d75aa ...
# Packet no:           2, Packet size:         102, Data: c348138e ...
# ----------------------------------------------------------------
# -- TRANSACTION TABLE
# ----------------------------------------------------------------
# Size:           1
# Items added:           1
# Items removed:           0
# Packet no:           0, Packet size:           4, Data: 00000000
# Packet no:           1, Packet size:          42, Data: b90d75aa ...
# Packet no:           2, Packet size:          94, Data: 5ebfc4eb ...
# ----------------------------------------------------------------
```

V okně Transcript se můžeme dále dozvědět jak se testem podřilo splnit function coverage.

Pokud se něco nedaří zkontrolujte zda je nainstalovaná nejnovější verze nástroje Modelsim.