



POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Praca dyplomowa magisterska

Sprzętowy system analizy i monitorowania ruchu sieciowego

autor: Łukasz Nowok

kierujący pracą: dr hab. inż. Adam Ziębiński

Gliwice, listopad 2019

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 6 listopada 2019

.....
(podpis)

.....
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Oświadczenie promotora

Oświadczam, że praca „Sprzętowy system analizy i monitorowania ruchu sieciowego” spełnia wymagania formalne pracy dyplomowej magisterskiej.

Gliwice, dnia 6 listopada 2019

.....
(podpis promotora)

Spis treści

1	Wstęp	1
1.1	Cel pracy	1
1.1.1	Zakres pracy	2
2	Analiza tematu	3
2.1	Ethernet	3
2.1.1	Proces wysłania i odebrania ramki Ethernet	4
2.1.2	RGMII	6
2.1.3	MDIO	7
2.2	Sposoby nasłuchiwanego ruchu sieciowego	8
2.2.1	Regenerujący koncentrator	9
2.2.2	Przełącznik sieciowy	9
2.2.3	Komputer z dwu-portową kartą sieciową	10
2.2.4	Dedykowane rozwiązanie typu TAP	11
2.2.5	Systemy oparte na układzie FPGA	12
2.3	Analiza ruchu sieciowego	13
2.3.1	Analiza offline - programy	14
2.4	Układ FPGA	18
2.5	Protokół PCI-Express	19
2.5.1	Dostępne rozwiązania na układy FPGA	19
2.6	Wykorzystane narzędzia	20
2.6.1	Karta deweloperska NetFPGA-1G-CML	20
2.6.2	Środowisko Vivado	22
2.6.3	Bloki IP Core	23

2.6.4	Protokół AXI4-Stream	24
2.6.5	Visual Studio Community	25
2.6.6	Python	25
2.6.7	Trafgen	26
3	Przedmiot pracy	27
3.1	Opis systemu	28
3.1.1	Zasada działania	29
3.1.2	Mozliwe konfiguracje	29
3.1.3	Ograniczenia teoretyczne	30
3.1.4	Plan prac	31
3.2	Część sprzętowa	33
3.2.1	Przechwytywanie	35
3.2.2	Pakietyzacja ramek	37
3.2.3	Filtracja	40
3.2.4	Blok MDIO	48
3.2.5	Blok kontrolujący	48
3.2.6	Wymiana danych	50
3.2.7	Wyniki implementacji projektu sprzętowego	52
3.3	Część programowa	53
3.3.1	Uzasadnienie wyboru języka programowania oraz interfejsu .	53
3.3.2	Komunikacja z blokiem Xillybus	54
3.3.3	Przechwytywanie - <i>capture</i>	55
3.3.4	Nastawy filtrów - <i>set</i>	56
3.3.5	Sterowanie tunelami - <i>bridge</i>	58
3.3.6	Rejestry MDIO - <i>mdio</i>	59
3.3.7	Opcja EEE - <i>eee_off</i>	59
3.4	Uruchomienie i testowanie	59
3.4.1	Testowanie	62
3.4.2	Wpływ trybu Energy-Efficient Ethernet na opóźnienia	66
4	Badania	69
4.1	Analiza zdolności przechwytywania	69

4.1.1	Stanowisko badawcze	69
4.1.2	Metodyka pomiarowa	70
4.1.3	Wyniki	71
4.2	Eksperyment precyzji pomiarowej	72
4.2.1	Stanowisko badawcze	72
4.2.2	Metodyka pomiarowa	73
4.2.3	Wyniki	74
4.3	Analiza czasu filtracji	75
4.3.1	FPGA	75
4.3.2	CPU	76
4.3.3	Wyniki	80
5	Podsumowanie	85

Rozdział 1

Wstęp

Obecne sieci komputerowe oparte na standardzie Ethernet IEEE 802.3 są dużo szybsze i niezawodne, niż miało to miejsce kilkanaście lat temu. Obserwuje się nieustanny wzrost prędkości łącza, spowodowany coraz to większym zapotrzebowaniem na szybkość przesyłania danych. Cechy Ethernetu, takie jak: prostota, niewielkie koszty, niezawodność, łatwość instalacji i rozbudowy, przyczyniły się do tego, że jest to teraz najpopularniejsze rozwiązanie komunikacyjne. [21].

Zwiększanie prędkości standardu Ethernet prowadzi do sytuacji, gdzie przechwytywanie ruchu sieciowego za pomocą komputera klasy PC z podstawową kartą sieciową nie jest w stanie zapewnić odpowiedniej dokładności oraz niezawodności. Utrata ramek przy większym obciążeniu łącza jest bardzo prawdopodobna.

1.1 Cel pracy

Celem pracy było zaprojektowanie i zbadanie systemu pozwalającego na monitorowanie oraz wstępna analizę ruchu sieciowego. Do tego celu należało wykorzystać kartę NetFPGA-1G-CML dostępną na wyposażeniu Instytutu Informatyki.

Główne założenia projektu:

- transparentność dla istniejącej sieci,
- monitoring nie wprowadzający znaczących opóźnień,
- sprzętowa analiza danych,

- zapis do pliku,
- brak utraty pakietów, nawet przy maksymalnej przepustowości oraz liczby ramek na sekundę,
- sprzętowe znaczniki czasu przechwycenia ramki.

1.1.1 Zakres pracy

Praca wymaga zapoznania się z takimi elementami jak:

- Układy FPGA, język opisu sprzętu i ich możliwości;
- Standard Ethernet IEEE 802.3;
- PCI-Express;

Kolejno praca wymagała:

- Zaprojektowania programu sprzętowego na układ FPGA, który ma na celu monitoring oraz analizę ruchu sieciowego;
- Napisania aplikacji systemowej do obsługi programu sprzętowego;
- Porównania systemu z popularnymi rozwiązaniami programowymi;

Rozdział 2

Analiza tematu

Temat sprzętowego systemu monitorowania i analizy ruchu sieciowego wiąże się z zagadnieniami takimi jak:

- Ethernet,
- Metody monitorowania i analizy ruchu sieciowego,
- Układy FPGA,
- PCI Express.

W tym rozdziale zostaną poruszone wyżej wymienione tematy, które są potrzebne w zrozumieniu zagadnienia oraz dokumentujące wybraną ścieżkę przy projektowaniu rozwiązania sprzętowego. Zostaną wskazane ogólne wady i zalety poszczególnych rozwiązań. Opisane zostaną także narzędzia wykorzystanie podczas tworzenia i testowania systemu.

2.1 Ethernet

Ethernet jest wykorzystywany do budowy różnych wielkości sieci. Standard ten obejmuje specyfikację przewodów oraz przesyłanych nimi sygnałów. Opisuje format ramek i protokoły dwóch najniższych warstw modelu ISO/OSI (rys. 2.3). Standard Ethernet jest utrzymywany i rozwijany przez Instytut Inżynierów Elektryków i Elektroników (ang. *Institute for Electrical and Electronics Engineers - IEEE*). Dokładna specyfikacja jest zawarta w dokumentach IEEE 802.3 [11].

Ethernet rozwija się bardzo szybko. Obecnie standardem w zastosowaniach domowych stają się urządzenia wspierające prędkość transmisji ramek na poziomie 1 Gbit/s, a powoli zaczynają pojawiać się urządzenia obsługujące prędkości od 2.5 Gbit/s do 10 Gbit/s. W zastosowaniach komercyjnych i naukowych istnieją już rozwiązania wspierające technologię przesyłu z prędkością nawet 400 Gbit/s, które wchodzą w skład urządzeń z grupy terabitowego Ethernetu.

Szybki rozwój spowodował, że standard Ethernet stosowany jest nie tylko w dostarczaniu usług dostępu do Internetu, ale także w przemyśle, lotnictwie [7] oraz motoryzacji [12]. Używany jest do komunikacji pomiędzy różnego rodzaju czujnikami i sterownikami. Takie zastosowania charakteryzują się dużo większymi ostrzeniami od strony sprzętu. Jednym z przykładów może być wykorzystywanie protokołu PTP (ang. *Precision Time Protocol*), który umożliwia w odpowiedniej konfiguracji sieci synchronizację czasu urządzeń na poziomie kilkuset nanosekund.

2.1.1 Proces wysłania i odebrania ramki Ethernet

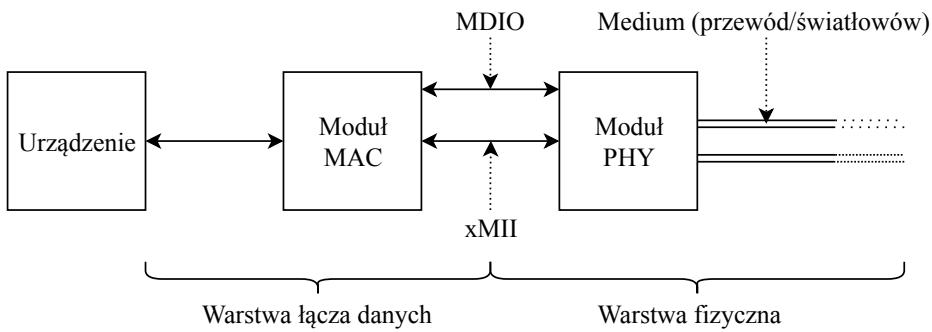
Ramka Ethernet złożona jest z segmentów przedstawionych na rysunku 2.1. Preambuła składa się z 8 bajtów naprzemiennego ciągu jedynek i zer (101010...), która zakończona jest podwójną jedynką (..101011). Podwójna jedynka oznacza początek głównej ramki. Kolejno znajdują się adres docelowy i adres źródłowy MAC (ang. *Media Access Control Address*). Następne 2 bajty określają długość lub typ ramki. Jeśli wartość w tym polu jest mniejsza niż 1536 to określa długość ramki, a jeżeli większa lub równa to typ protokołu. Kolejno znajdują się dane, których rozmiar jeżeli jest mniejszy niż 46 bajtów to uzupełniane są zerami. Ostatnim polem jest suma kontrolna FCS (ang. *Frame Check Sequence*).

8 bajtów	6 bajtów	6 bajtów	2 bajty	46-1500 bajtów	4 bajty
Preambuła	Adres Docelowy	Adres Źródłowy	Typ/ Długość	Dane	CRC

Rysunek 2.1: Podstawowa ramka Ethernet.

Proces wysłania/odebrania ramki Ethernet przez dane medium wymaga ko-

munikacji z układem PHY. Modułem pośredniczącym w tym procesie jest układ MAC (ang. *Media Access Control*). Na rysunku 2.2 znajduje się blokowy diagram tego procesu.



Rysunek 2.2: Schemat połączenia urządzenia do sieci Ethernet.

Komunikacja z modułem MAC może być zapewniona na różne sposoby. Jeżeli urządzeniem jest komputer klasy PC, to najczęściej do komunikacji wykorzystywane jest połączenie PCI-Express. Cały proces obudowany jest w takim przypadku sterownikami i stosem sieciowym systemu operacyjnego. Urządzeniem wysyłającym ramkę ethernetową może być także mikrokontroler. W tej sytuacji komunikacja najczęściej odbywa się po interfejsie SPI (ang. *Serial Peripheral Interface*). Moduł MAC kolejno komunikuje się z modułem PHY. Tutaj wykorzystywany jest interfejs z grupy xMII (ang. *Media-Independent Interface*, np. RGMII, MII). Przy wysyłaniu ramki moduł MAC jest odpowiedzialny za dodanie preambuły i odpowiednio długiej minimalnej przerwy pomiędzy ramkami. Często, aby odciążyć główny procesor także za dodanie adresu źródłowego i wyliczenia sumy kontrolnej. Przy odbieraniu danych sprawdza integralność danych, opcjonalnie odrzuca ramki z niepoprawnym adresem MAC. Następnie moduł PHY zamienia strumień danych na sygnały elektryczne bądź optyczne. Oprócz tego negocjuje prędkość połączenia z układem PHY urządzenia docelowego. Moduł PHY posiada szereg rejestrów, które pozwalają na odpowiednią konfigurację. Do tego celu używany jest osobny interfejs MDIO (ang. *Management Data Input/Output*).

Na rynku dostępne są także układy hybrydowe, które łączą ze sobą funkcje MAC i PHY w jednym module.

2.1.2 RGMII

RGMII (ang. *Reduced Gigabit Media Independent Interface*) to zredukowana postać interfejsu GMII, która służy do połączenia układu PHY z układem MAC. Zamiast standardowych 28-pinów wykorzystuje wyłącznie 12-pinów. Redukcja liczby pinów jest możliwa poprzez zmianę sposobu przekazywania danych z SDR (ang. *single data rate*) na DDR (ang. *double data rate*). Oznacza to, że stan logiczny sygnałów zatrzaszkowany jest przy zboczu narastającym i opadającym zegara. Tryb DDR używany jest tylko w przypadku najwyższej prędkości 1000 Mbit/s. Częstotliwość zegara interfejsu jest zależna od wynegocjowanej prędkości linka przez układ PHY i wynosi 125 MHz, 25 MHz i 2.5 MHz dla odpowiednio prędkości 1000 Mbit/s, 100 Mbit/s i 10 Mbit/s.

Sygnały interfejsu można podzielić na sygnały generowane oraz odbierane przez układ PHY:

- Sygnały generowane przez układ PHY:
 - RXC - zegar referencyjny 125 MHz, 25 MHz, 2.5 MHz zależny od prędkości połączenia.
 - RD[3:0] - 4-bitowa szyna danych zawierająca dane ramki odbieranej. Na zboczu narastającym zegara RXC zawiera bity 3:0, a na zboczu opadającym 7:4 odbieranego bajtu ramki.
 - RX_CTL - sygnał kontrolny. Na zboczu narastającym zegara RXC zawiera sygnał RXDV, a na zboczu opadającym RXERR.
- Sygnały wysyłane do układu PHY:
 - TXC - zegar referencyjny 125 MHz, 25 MHz, 2.5 MHz zależny od prędkości połączenia.
 - TD[3:0] - 4-bitowa szyna danych zawierająca dane ramki wysyłanej. Na zboczu narastającym zegara TXC zawiera bity 3:0, a na zboczu opadającym 7:4 wysyłanego bajtu ramki.
 - TX_CTL - sygnał kontrolny. Na zboczu narastającym zegara RXC zawiera sygnał TXDV, a na zboczu opadającym TXERR.

Ciekawą funkcją interfejsu jest kodowanie informacji o stanie połączenia podczas przerwy między odbieranymi ramkami. Kiedy sygnały RXDV oraz RXERR są w stanie logicznego zera, na sygnatach RD[3:0] kodowane są takie informacje jak status połączenia, prędkość połączenia, czy tryb pracy half-duplex/full-duplex.

W przypadku zegara RXC należy pamiętać o tym, że nie jest on przesunięty w fazie o 90 stopni w stosunku do pozostałych danych. Oznacza to, że w części układu odbierającego wymagane jest dodanie odpowiedniego opóźnienia lub mechanizmu, który skoryguje zegar.

Pełna specyfikacja interfejsu znajduje się w dokumencie [8].

2.1.3 MDIO

MDIO (ang. *Management Data Input/Output*) to magistrala szeregową wykorzystywana do zarządzania układami PHY. Interfejs działa na zasadzie mechanizmu Master-Slave. Na pojedynczej linii magistrali wspiera jedno urządzenie typu Master, które przeprowadza transakcje oraz do 32 urządzeń typu Slave, które nasłuchują i odpowiadają. Połączenie składa się z dwóch sygnałów:

- MDC - sygnał zegarowy o minimalnym okresie 400 ns, czyli 2.5 MHz (w nowszych układach PHY częstotliwość może być wyższa), generowany przez urządzenie Master;
- MDIO - sygnał współdzielony przez wszystkie urządzenia do transmisji danych;

Pakiet składa się z dwóch 32-bitowych części. Pierwsza z nich to preambuła składająca się z samych logicznych jedynek, jednak najczęściej w układach PHY domyślnie załączona jest opcja *preamble suppression*, która pozwala na pominięcie tej części. Na kolejne 4 bajty składają się takie pola, jak:

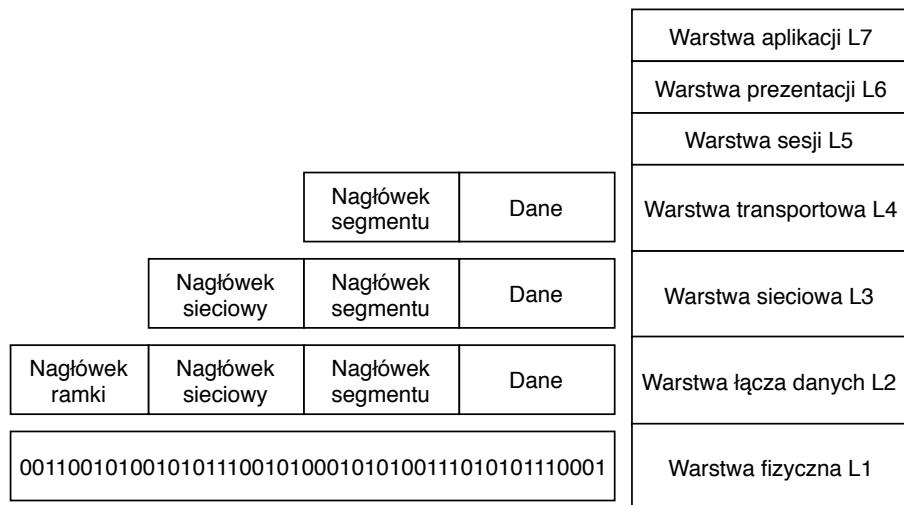
- ST - 2-bitowe pole startu transakcji, które zawsze ma wartość 01;
- OP - 2-bitowe pole określające zapis (wartość 01) lub odczyt (wartość 10);
- PA5 - 5-bitowy adres układu PHY;
- RA5 - 5-bitowy adres rejestru;

- TA - 2 wolne bity pozwalające na przełączenie się układu PHY w tryb nadawania;
- D15 - 16-bitowe pole zawierające odbierane lub nadawane dane;

Wartości adresów oraz rejestrów poszczególnych funkcji układów PHY są opisane w dokumentacji producenta danego układu PHY.

2.2 Sposoby nasłuchiwania ruchu sieciowego

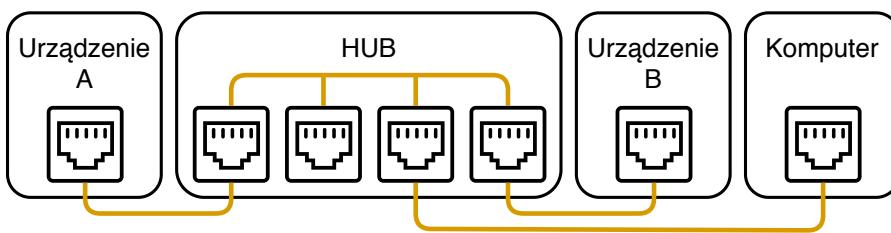
Istnieje wiele możliwości nasłuchiwania połączenia sieciowego. Do tego celu można skorzystać z rozwiązań programowych jak i w pełni sprzętowych [1]. Wybór rozwiązania zależy od postawionych wymagań i stawianych mu ograniczeń. Cel może wymagać rozwiązania w pełni przezroczystego dla istniejącej infrastruktury sieciowej lub/i wprowadzającego jak najmniejsze opóźnienia. Wymaganiem może też być archiwizacja, filtrowanie pakietów w locie lub precyzyjny zapis czasu odebrania ramki. Przyjętym w pracy ograniczeniem jest brak możliwości konfiguracji urządzeń końcowych, pomiędzy którymi nasłuchiwanym jest ruch sieciowy. Takie ograniczanie eliminuje wykorzystanie oprogramowania na samych urządzeniach co skutkuje, że monitorowanie musi odbywać się na najniższych poziomach modelu ISO/OSI.



Rysunek 2.3: Model ISO/OSI.

2.2.1 Regenerujący koncentrator

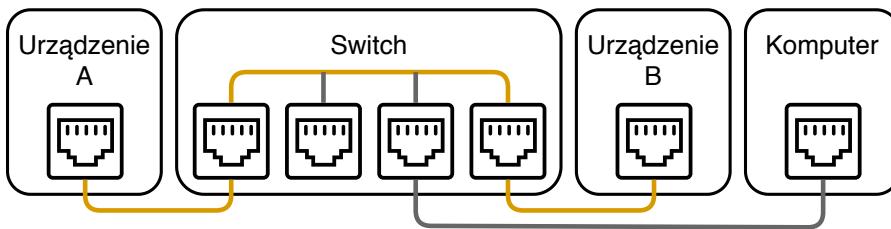
W sieciach Ethernet o prędkości do 100 Mbit/s jedną z możliwości jest zastosowanie pasywnego regenerującego koncentratora (ang. *Passive Repeating HUB*). Ze względu na swoją budowę sygnał na każdym z portów jest identyczny. Takie rozwiązanie pozwala nasłuchiwać cały ruch sieciowy w obrębie danej sieci. W tym przypadku wystarczy podłączenie komputera zainstalowanym oprogramowaniem do zbierania pakietów pod dowolny wolny port. Jedną z wad jest to, że sieci oparte o regenerujące koncentratory obecnie już nie występują z powodu swoich ograniczeń. Rozwiązanie to działa na warstwie fizycznej modelu ISO/OSI.



Rysunek 2.4: Schemat monitoringu z wykorzystaniem regenerującego koncentratora.

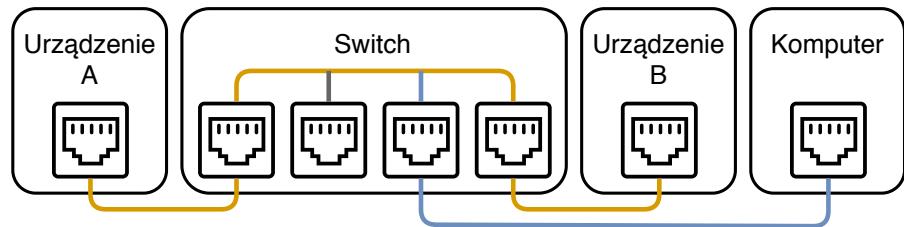
2.2.2 Przełącznik sieciowy

Najpopularniejszym rozwiązaniem podłączenia w sieć kilku urządzeń jest wykorzystanie przełącznika sieciowego (ang. *Switch*). W tym przypadku nasłuchiwanie transmisji z dowolnego wolnego portu nie jest możliwe, ponieważ ruch sieciowy pomiędzy dwoma urządzeniami jest niewidoczny dla innych urządzeń. Wszystkie ramki o danym adresie MAC są przekazywane prosto do docelowego urządzenia o takim adresie. Rozwiązanie to działa na warstwie łącznika danych modelu ISO/OSI.



Rysunek 2.5: Schemat połączenia przełącznika sieciowego.

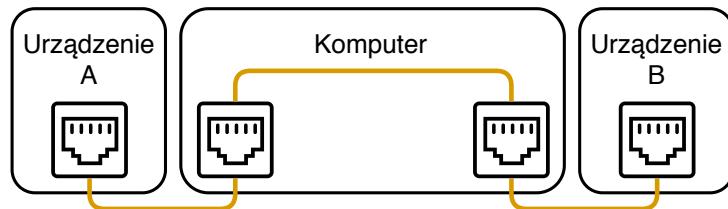
Do celów diagnostycznych producenci przełączników sieciowych często implementują funkcję lustrzanego odbicia portu (ang. *port mirroring*) lub dedykowany port służący do tego celu. Opcja ta dostarcza możliwość skonfigurowania przełącznika w taki sposób, aby ruch sieciowy z danego portu lub portów był klonowany i przekazywany na wskazany inny port. Wadą rozwiązania jest potrzeba rekonfiguracji przełącznika sieciowego oraz to, że w przypadku dużego natężenia ruchu sieciowego możliwa jest utrata pakietów. Rozwiążanie to działa na warstwie łącza danych modelu ISO/OSI.



Rysunek 2.6: Schemat monitoringu z użyciem przełącznika oraz funkcji port mirroring'u.

2.2.3 Komputer z dwu-portową kartą sieciową

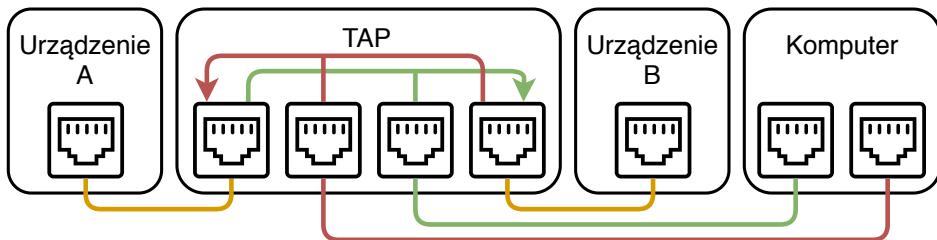
Kolejnym rozwiązaniem jest wykorzystanie komputera pośredniczącego w połączeniu. Odpowiednie oprogramowanie transportuje ramki z jednego portu na drugi tworząc tunel. Taka konfiguracja umożliwia podgląd ruchu sieciowego oraz dodatkowo pozwala na ingerencję w transmisję. Wadą jest narzut systemu operacyjnego na komputerze, który może eliminować takie zastosowanie w sytuacjach, gdzie wymagane są niskie opóźnienia oraz deterministyczne zachowanie. Utrata pakietów podczas większego użycia łącza staje się bardzo prawdopodobna.



Rysunek 2.7: Schemat monitoringu typu man-in-the-middle.

2.2.4 Dedykowane rozwiązanie typu TAP

Najlepszą z możliwości jest zastosowanie urządzenia typu TAP (ang. *Test Access Point*). Urządzenie to posiada dwa porty, które tworzą tunel. Cały ruch sieciowy jest przekazywany poprzez wybrany przez producenta interfejs do użytkowania. Najczęściej jest to podwójny port o prędkości wyższej lub równej prędkości portów monitorowanych. Na jednym z portów znajdują się dane pochodzące z drogi A do B, a na drugim z B do A. Istnieją urządzenia które zamiast przekazywać ruch po portach Ethernet, przesyłają dane po magistrali USB do komputera. Zaletą korzystania z TAP'ów jest przezroczysty sposób ich działania. Urządzenia podsłuchiwane nie są w stanie wykryć narzędzia pośredniczącego. Dodatkowo, taki rodzaj instalacji wprowadza niewielkie opóźnienia oraz nie ogranicza przepustowości. Rozwiązanie to działa na warstwie fizycznej modelu ISO/OSI.



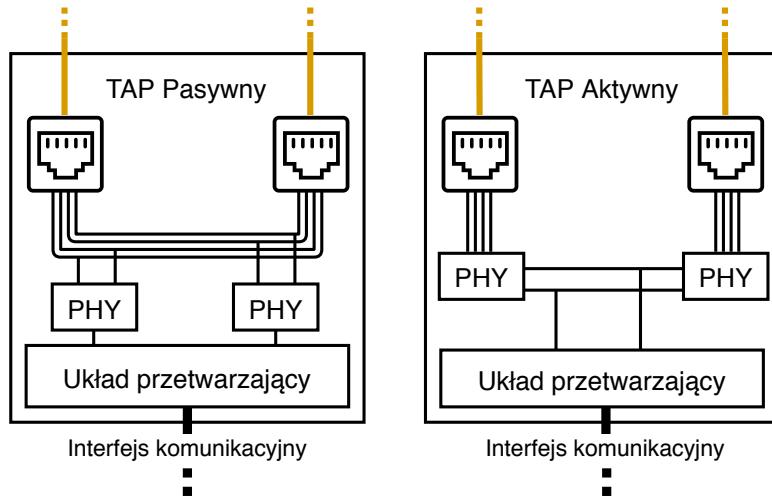
Rysunek 2.8: Schemat monitoringu z wykorzystaniem urządzenia typu TAP.

Urządzenia typu TAP

Urządzenia tego typu można podzielić na dwa rodzaje: pasywne oraz aktywne. Pasywny rodzaj można zastosować w przypadku, kiedy połączenie posiada niezależne przewody do odbierania i wysyłania. Przykładem mogą być urządzenia działające w technologiach 10/100Base-T lub światłowodowych.

Inaczej jest w standardzie 1000Base-T (IEEE 802.3ab), gdzie transmisja odbywa się poprzez wszystkie pary skrętki FTP jednocześnie. Dane przechodzące w obu kierunkach nachodzą na siebie i wzajemnie się modulują. Układ PHY na końcu połączenia zna informację, którą w danym momencie wysyła, dlatego jest w stanie odkodować transmisję przychodzącą. Bez tej informacji nie da się zrekonstruować komunikacji (na podstawie samej znajomości sygnałów elektrycznych), dlatego nie jest możliwe zastosowanie pasywnego TAP'a. W tym przypadku należy

skorzystać z TAP'a aktywnego. Uogólniony schemat działania obydwu rodzajów znajduje się na rysunku 2.9.



Rysunek 2.9: Blokowy model działania pasywnego i aktywnego urządzenia monitorującego.

Na rysunku 2.9 można zauważyć, że rodzaj pasywnego urządzenia w żaden sposób nie ingeruje w połoczenie. Nasłuchiwanie odbywa się wyłącznie poprzez analizę sygnałów elektrycznych lub optycznych. W przypadku, gdy taka analiza nie jest możliwa przez standard transmisji, wykorzystuje się typ aktywny. W tym przypadku jedno połoczenie jest rozdzielane na dwa. Transmisja przychodząca jest dekodowana przez moduł PHY na sygnały typu xMII możliwe do zinterpretowania przez układ przetwarzający. Kolejno ramki trafiają do przeciwnego modułu PHY, który transmituje ramki dalej. Takie rozwiązanie wprowadza pewne niewielkie opóźnienia do kilkuset nanosekund [26] (w przypadku łącza w standardzie 1000Base-T).

2.2.5 Systemy oparte na układzie FPGA

W środowisku badawczym spotykany rozwiązańem są systemy oparte o układy FPGA. Takie systemy zapewniają dużo większą elastyczność, niż dedykowane rozwiązania. Ze względu na specyfikę układów FPGA, platforma nie jest przeznaczona

wyłącznie do jednego celu. Możliwość tworzenia i modyfikacji projektów sprzętowych otwiera drogę do prototypowania i sprawdzania nowych rozwiązań.

Przykładem wykorzystania, może być analizator opóźnień wprowadzanych przez testowane urządzenie sieciowe [3]. Autorzy projektu stworzyli dany system z wykorzystaniem platformy NetFPGA. Innym przykładem może być także TAP do deterministycznego Ethernetu dla sieci 10/100BASE-T [6]. Tutaj autorzy użyli platformy Xilinx SP605 oraz łączą w standardzie 1000BASE-T do przekazywania danych.

Takie rozwiązania są zazwyczaj tańsze od specjalistycznych narzędzi, jednak wiąże się z tym potrzeba stworzenia projektu sprzętowego, którego proces jest dość skomplikowany i pracochłonny.

2.3 Analiza ruchu sieciowego

Analiza ruchu sieciowego to przetwarzanie informacji offline lub online nasłuchiwanego połączenia lub połączeń. Jest „zadaniem skomplikowanym ze względu na fakt bardzo dużej liczby danych, które muszą zostać przetworzone” [2]. Co więcej, taki proces jest trudny do zautomatyzowania. W większości przypadków wykorzystywana do tego celu jest wiedza eksperta. Proces taki można zautomatyzować w przypadku z góry określonego celu, jak np. analiza czasu pomiędzy określonymi pakietami, dzięki której można stwierdzić czy dane urządzenie lub sieć działa zgodnie z założeniami. Analiza ruchu sieciowego może poruszać bardzo wiele aspektów:

- przeszukiwanie zbiorów danych i wyszukiwanie relacji [2],
- analiza czasowa połączeń, np. pomiar opóźnień [3],
- analiza ruchu na poziomie warstwy fizycznej: utrata pakietów, stabilność, przepustowość,
- analiza danych specyficznych dla danego protokołu,
- analiza statystyczna ruchu sieciowego [4].

Ze względu na dużą różnorodność możliwych ścieżek rozwoju, postanowiono w części sprzętowej zaimplementować mechanizm filtracji, który pozwoli na znaczną redukcję danych. Filtracja pozwala na zapisanie wyłącznie interesujących nas ramek np. o danym adresie MAC, IP lub określonych wartościach poszczególnych bajtów w ramce. Mniejszy rozmiar pliku może znaczco przyspieszyć wykonywanie dalszej analizy offline. Zmniejszona ilość danych wspomagać może także analizę online wykonywaną programowo na komputerze.

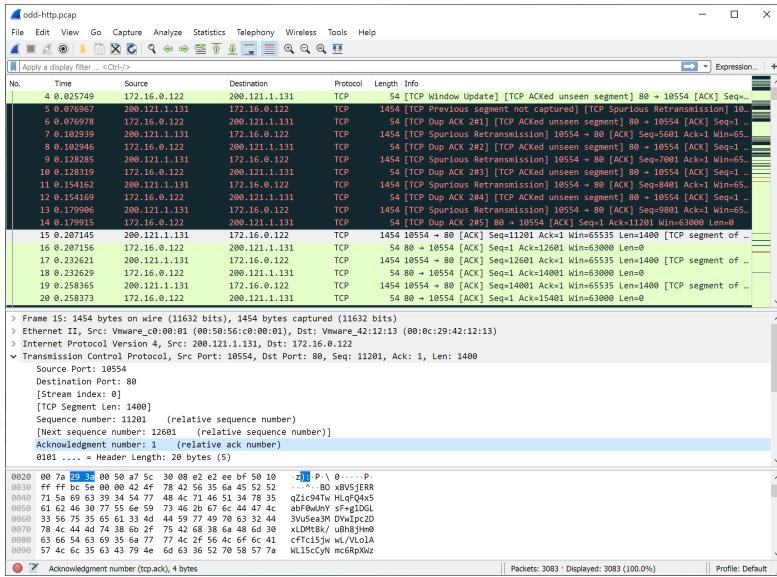
2.3.1 Analiza offline - programy

Przechwycony fragment ruchu sieciowego można przetwarzać z wykorzystaniem specjalizowanych do tego celu programów.

Wireshark

Jednym z najpopularniejszych programów w tej dziedzinie jest Wireshark [27]. Oprogramowanie posiada takie możliwości, jak:

- Głęboka inspekcja setek protokołów,
- Zbieranie pakietów w czasie pracy programu,
- Multiplatformowość - praca na systemach: Windows, Linux, OS X, FreeBSD, NetBSD i wiele innych,
- Wizualizacja danych sieciowych,
- Bogate opcje filtrowania,
- i wiele innych.



Rysunek 2.10: Zrzut ekranu z programu Wireshark

Tcpdump

Innym otwarto-źródłowym oprogramowaniem jest konsolowy program o nazwie *tcpdump*, który jest domyślnie zainstalowany na wielu dystrybucjach Linuxa. Posiada możliwości przechwytywania ruchu sieciowego i jego filtracji. Niestety w przypadku systemu Windows nie ma oficjalnie dostępnej wersji i wymagana jest komplikacja kodu źródłowego.

Format plików

Programy do analizy offline obsługują wiele różnych formatów plików. Dane zebrane za pomocą karty FPGA należy zapisać do formatu, który odczyta program. Najczęściej wykorzystywany formatami zapisu przechwytywanych danych są rozszerzenia *.pcap* oraz *.pcapng*.

Założenia:

- zapis znaczników w rozdzielcości nanosekund,
- jak najmniejsza modyfikacja strumienia danych od układu FPGA, aby zmimimalizować narzut procesora na konwersje,

- prosta konstrukcja pliku,
- możliwość zapisu całej ramki z warstwy łącza danych.

Format PCAP

Format PCAP jest bardzo prostym formatem plików. Według specyfikacji [24] spełnia wszystkie założenia. Nagłówek pliku składa się z takich parametrów, jak:

- Magiczna liczba (ang. *magic number*) - jest to liczba pozwalająca na detekcję systemu zapisu liczb większych niż 1 bajt (Little-endian lub Big-endian) oraz detekcję rozdzielczości znaczników czasowych (milisekundy lub nanosekundy).
- Numer wersji major.minor - wersja formatu pliku (domyślnie: 2.4).
- Korekcja czasu GMT - wartość czasu w sekundach pomiędzy czasem GMT, a czasem lokalnym (domyślnie: 0).
- Dokładność czasu - wartość określająca rozdzielczość, w praktyce nieużywana (domyślnie: 0).
- Maksymalna długość - wartość określająca możliwą maksymalną długość ramki (domyślnie: 65535).
- Typ linku - identyfikator formatu zapisywanych danych [23].

Tablica 2.1: Struktura nagłówka formatu pliku PCAP.

Nazwa\Numer bajtu	1	2	3	4
Magiczna liczba	0x4D3CB2A1			
Wersji pliku (2.4)	0x0002	0x0004		
Korekcja czasu GMT	0x00000000			
Dokładność czasu	0x00000000			
Maksymalna długość w bajtach	0x0000FFFF			
Typ linku (LINK_TYPE)	0x00000112			

Po nagłówkach znajdują się dane w strukturze składającej z nagłówka ramki oraz samej ramki. Nagłówek ramki składa się z pól:

- Sekundowy znacznik czasowy (32bit) - zawiera liczbę sekund od czasu 1 Stycznia 1970 roku 00:00:00 GMT (znany jako czas UNIX'owy).
- Mikrosekundowy lub nanosekundowy znacznik czasowy (32bit) - zawiera precyzyjniejszą informację o częściach sekundy i razem z powyższą wartością tworzy całość.
- Sumaryczna długość danych (32bit) - liczba zapisanych w pliku bajtów danego pakietu.
- Rzeczywista liczba bajtów ramki (32bit) - ilość rzeczywistych danych wysłanych lub odebranych z interfejsu; ta wartość zawsze jest mniejsza lub równa sumarycznej długości danych.

Format PCAPNG

Format *PCAPNG* (ang. PCAP New Generation) jest nowszą wersją formatu plików *PCAP*. Posiada więcej możliwości w porównaniu do poprzednika. Według specyfikacji [16] format pozwala na zapis danych z kilku interfejsów, ich identyfikację, dodawania komentarzy, nazw i wiele więcej. Struktura pliku znacząco się różni od poprzedniej wersji. Dodatkowe możliwości powodują, że narzut na rozmiar pliku wynikowego staje się większy. Nagłówek dla każdej ramki posiadający wymagane możliwości jest większy o 16 bajtów od formatu *PCAP*.

Ostateczny format zapisu

Obydwa formaty plików spełniają wszystkie postawione założenia. Formaty korzystają z tego samego identyfikatora typu linka. Wartość identyfikatora pozwala na zidentyfikowanie formatu zapisanych w pliku danych. Rodzaj LINK-TYPE_ETHERNET_MPACKET (0x0112) określa, że w danych występuje cały ciąg ramki ethernetowej zaczynając od preambuły, a kończąc na sumie kontrolnej.

Należało podjąć decyzję, który format plików będzie wykorzystywany w projekcie. Zdecydowano się na format *.pcap* ze względu na mniejszy rozmiar wynikowy pliku oraz prostszą strukturę. Dzięki prostej strukturze, pakietyzacja ramek

na układzie FPGA jest mniej skomplikowana. Strumień danych otrzymywany z układu FPGA nie wymaga dodatkowego parsowania przez program. Wadą jest brak możliwości rozróżnienia interfejsów, jednakże problem można rozwiązać zapisując strumień danych z interfejsów w osobnych plikach.

2.4 Układ FPGA

Układ FPGA (ang. *field-programmable gate array*) to bezpośrednio rekonfigurowalna macierz bramek. Układ umożliwia stworzenie dowolnej cyfrowej funkcji. Projektowanie układu cyfrowego z wykorzystaniem FPGA w dużym uproszczeniu polega na ustaleniu odpowiednich połączeń pomiędzy wbudowanymi w układ blokami. Połączenia mogą być odpowiednio zmieniane i modyfikowane bez ograniczeń ilościowych. [20]

Układ FPGA zapewnia kompromis pomiędzy kosztami produkcji specjalizowanego układu scalonego, czasem potrzebnym do stworzenia projektu, a kosztem samego układu. FPGA jest układem znacznie droższym od układu ASIC biorąc pod uwagę wyłącznie jednostkowy koszt produkcji, jednakże w projektach małonakładowych koszt stworzenia linii produkcyjnej dedykowanego układu przewyższa cenę zastosowania FPGA.

Proces tworzenie projektów na układy FPGA polega na opisaniu sprzętu. Do tego celu wykorzystuje się języki Verilog lub VHDL. Producenci układów dostarczają środowiska przystosowane do projektowania programów sprzętowych. Firma Xilinx do swoich najnowszych produktów oferuje oprogramowanie Vivado, które udostępnia możliwość przeprowadzenia całego procesu od projektowania, symulacji do syntezy, implementacji i programowania układów.

Zaletą układów FPGA jest szybkość przetwarzania informacji i efektywność energetyczna. Układ nie realizuje kolejnych instrukcji programu jak w przypadku procesora CPU, a przetwarza dane według stworzonego projektu, często równolegle. Taka struktura umożliwia przetwarzanie gigabajtów danych na sekundę, będąc przy tym oszczędna w aspekcie poboru prądu.

Taki układ powinien bardzo dobrze sprawdzić się w dziedzinie przechwytywania i przetwarzania ruchu sieciowego.

2.5 Protokół PCI-Express

PCI-Express to trzecia generacja interfejsu komunikacyjnego typu *I/O*. Jest to wysoko wydajne połączenie Point-to-Point wykorzystywane do połączenia urządzeń peryferyjnych w aplikacjach takich jak przetwarzanie danych i komunikacja [17]. Taki typ interfejsu jest szeroko stosowany w komputerach klasy PC, laptopach i serwerach do podłączenia różnego rodzaju kart rozszerzeń. Karty graficzne, akceleratory obliczeniowe, dyski NVMe, karty sieciowe, karty DAQ i wiele innych wykorzystują to połączenie do komunikacji z głównym procesorem. W zależności od wersji oferuje różne prędkości z których oficjalnie dostępne zostały przedstawione w tabeli 2.2.

Tablica 2.2: Przepustowość PCI Express [17].

Wersja	Kodowanie	Transfer	Przepustowość jednej linii
1.0b / 1.1	8b/10b	2.5 GT/s	2 Gbit/s
2.0 / 2.1	8b/10b	5 GT/s	4 Gbit/s
3.0 / 3.1	128b/130b	8 GT/s	7.9 Gbit/s
4.0	128b/130b	16 GT/s	15.8 Gbit/s

Połączenie PCI-Express jest pełnodupleskowe, czyli w obu kierunkach oferuje taką samą wydajność. Dodatkowo jedno połączenie może korzystać z większej liczby linii, jak 2, 4, 8, 12, 16 czy 32. Przy wykorzystaniu większej liczby linii proporcjonalnie zwiększa się maksymalna przepustowość.

2.5.1 Dostępne rozwiązania na układy FPGA

Firma Xilinx oferuje w pakiecie Vivado blok *7 Series FPGAs Integrated Block for PCI Express v3.3* służący do wykorzystania połączenia PCI-Express w procesie wymiany danych. Niestety dla mało doświadczonej osoby implementacja rozwiązania w projekcie może okazać się trudna i pracochłonna. Na rynku istnieją gotowe rozwiązania (tzw. wrapery), które obudowują wspomniany wcześniej blok w taki sposób, aby zastosowanie w projekcie było jak najprostsze oraz wydajne. Takimi rozwiązaniami są:

- Xillybus,

- RIFFA 2.2,
- JetStream,
- EPEE,
- ffLink,
- DyRACT.

Według [25] wszystkie rozwiązania zapewniają wysoką wydajność przewyższającą wymagania potrzebne w realizacji projektu. Wiele sprowadza się do wymagań funkcjonalnych oraz używanych przez rozwiązania zasobów sprzętowych. Jednym z wymogów projektowanego systemu jest działanie na platformie Windows, co zawiera możliwości do dwóch rozwiązań: Xillybus oraz RIFFA. W obydwu przypadkach dostępne są przykładowe projekty na platformy oparte o układy firm Xilinx oraz Altera. Oba rozwiązania dostarczają także dobrą dokumentację oraz sterowniki na systemy Windows i Linux. Zaletą RIFFA jest otwarto-źródłowość, niestety osiągnięcia odpowiedniej stabilności połączenia może wymagać dużego nakładu pracy [18]. Ostatecznie zdecydowano się na rozwiązanie firmy Xillybus, ponieważ udostępnia najprostszy w implementacji interfejs, umożliwia generację wyspecjalizowanego cora'a przez stronę internetową oraz zużywa dużo mniej zasobów sprzętowych niż rozwiązanie RIFFA [25].

2.6 Wykorzystane narzędzia

2.6.1 Karta deweloperska NetFPGA-1G-CML

Do realizacji projektu wykorzystana została karta deweloperska NetFPGA-1G-CML firmy Digilent. Karta oparta jest na układzie Xilinx Kintex-7 XC7K325T-1FFG676. Karta posiada 4 porty Ethernet w pracującym standardzie 1000/100/10BASE-T. Projekt NetFPGA to otwarta platforma do przeprowadzania badań i eksperymentów. Platforma składa się z karty oraz źródeł z referencyjnymi projektami[14].



Rysunek 2.11: Zdjęcie karty NETFPGA-1G-CML.

Główne cechy karty NetFPGA-1G-CML:

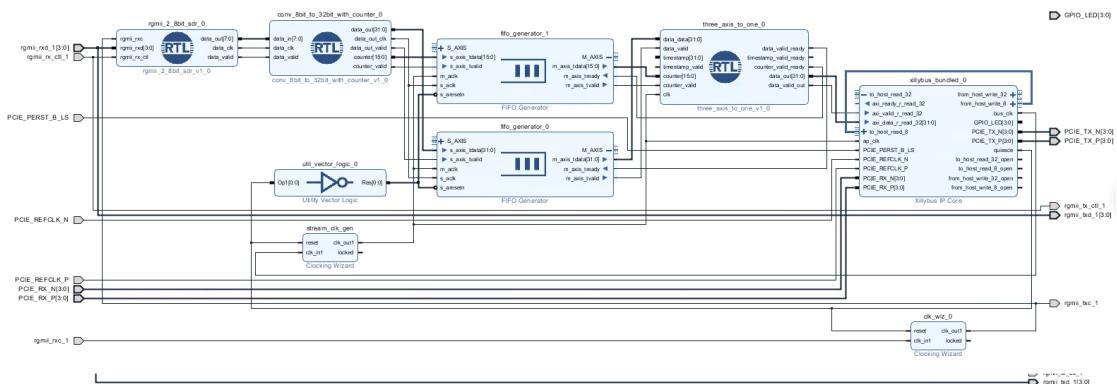
- Złącze PCI-Express 2.0 x4
- 1-Gigabitowa pamięć BPI Flash do przechowywania konfiguracji
- X16 4.5MB QDRII+ SRAM (450MHz)
- X8 512MB DDR3 DRAM (800Mhz)
- Slot na kartę SD
- Nisko-jitterowy 200MHz oscylator
- 4 porty Ethernet z układami PHY RTL8211E-VL z interfejsem RGMII
- Konektor FMC (VITA 57)
- 4 wskaźniki LED oraz 4 przyciski do dowolnego wykorzystania

Projekty z repozytorium NetFPGA wymagają wykorzystania systemu Fedora 20 wraz z oprogramowaniem ISE 14.6 i Vivado Design Suite 15.2. Niestety stworzenie takiego środowiska w czasie trwania prac nad projektem nie było możliwe.

Platforma NetFPGA bardzo dobrze nadaje się do nauki tworzenia urządzeń sieciowych.

2.6.2 Środowisko Vivado

Oprogramowanie to umożliwia projektowanie, symulowanie, syntezę, implementację oraz generację tzw. bitstreamów na układy FPGA firmy Xilinx. Tworzenie projektów może odbywać się na dwa sposoby. Pierwszym z nich jest tworzenie projektu w czystym Verilogu i/lub VHDL'u. Drugim sposobem jest wykorzystanie schematu blokowego *Block Design*, którego przykład został pokazany na rysunku 2.12.



Rysunek 2.12: Przykładowy schemat blokowy z programu Vivado.

Firma Xilinx oferuje w pakiecie wiele gotowych rozwiązań w postaci IP Core'ów. Oprócz tego możliwe jest tworzenie własnych bloków. Bloki można wygenerować opcją *Create and Package New IP*. Taki proces jednak jest dość pracochłonny. Przy modyfikacji plików źródłowych wymagany jest aktualizacja bloku, która składa się z wielu kroków. Inną opcją jest wykorzystanie bloków RTL, czyli automatycznie generowanych bloków na podstawie pliku źródłowego Verilog lub VHDL. Zaletą jest ich automatyczna generacja i aktualizacja po zapisaniu zmodyfikowanego pliku. Wadą, dużo mniejsze możliwości, jak np. brak okna konfiguracyjnego.

2.6.3 Bloki IP Core

Clocking Wizard

Blok *Clocking Wizard* upraszcza konwersję sygnałów zegarowych. Umożliwia generację sygnałów o różnej częstotliwości oraz wymaganych przesunięciach fazowych na podstawie podanego na wejściu sygnału zegarowego. Oprócz generacji kodu i obudowania komponentu *MMCM* lub *PLL* (w zależności od wprowadzonej konfiguracji) układu FPGA, definiuje także ograniczania i ramy czasowe wymagane do poprawnego działania modułu. Jeden blok jest w stanie wygenerować 7 różnych sygnałów zegarowych o różnych fazach i wypełnieniu. Konfigurator oferuje także wiele dodatkowych funkcji, których dokładny opis zawarty jest w dokumentacji [30].

AXI4-Stream FIFO

Blok buforujący *AXI4-Stream FIFO* jest pamięcią typu First-In-First-Out z interfejsem potokowym *AXI4-Stream*, który został opisany w rozdziale 2.6.4. W zależności od ustawionej w konfiguratorze szerokości szyny oraz głębokości kolejki blok wykorzystuje podstawowe komórki logiczne, jeżeli wymagana pamięć jest niewielka lub dedykowane komponenty *Block-RAM*, kiedy potrzebny jest duży bufor. Wspiera użycie niezależnych od siebie zegarów do odczytu i zapisu, dzięki czemu dobrze sprawdza się w aspekcie synchronizacji przesyłu danych w częściach projektu z różnymi sygnałami zegarowymi. Pełny opis funkcji znajduje się w dokumentacji [29].

Blok Xillybus - PCI-Express

W projekcie wykorzystano rozwiązanie firmy Xillybus, które upraszcza wykorzystanie połączenia PCI-Express, pomiędzy komputerem klasy PC, a kartą FPGA. Firma oferuje przykładowe projekty na wybrane modele zestawów deweloperskich, dobrą dokumentację oraz sterowniki na systemy Windows i Linux. Do pobrania dostępne są również wzorcowe programy z kodem źródłowym testujące podstawowe funkcje rozwiązania i przedstawiające poprawną obsługę interfejsu. Rozwiązanie dla celów niekomercyjnych i edukacyjnych jest bezpłatne z pewnymi

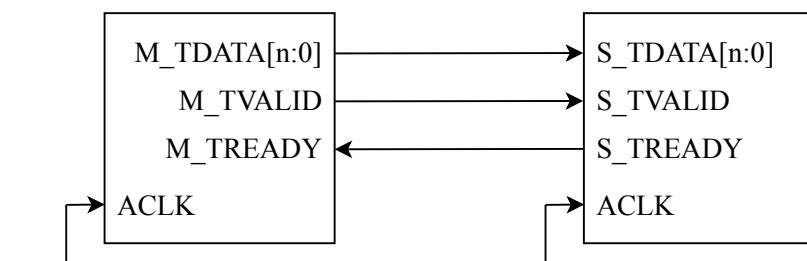
ograniczeniami funkcjonalnymi.

2.6.4 Protokół AXI4-Stream

Protokół *AXI4-Stream* jest protokołem strumieniowym. Protokół definiuje strony transakcji jako Master - nadawca oraz Slave - odbiorca. Interfejs posiada opcjonalny mechanizm kontroli przepływu danych. Najpopularniejsza odmiana składa się z trzech sygnałów:

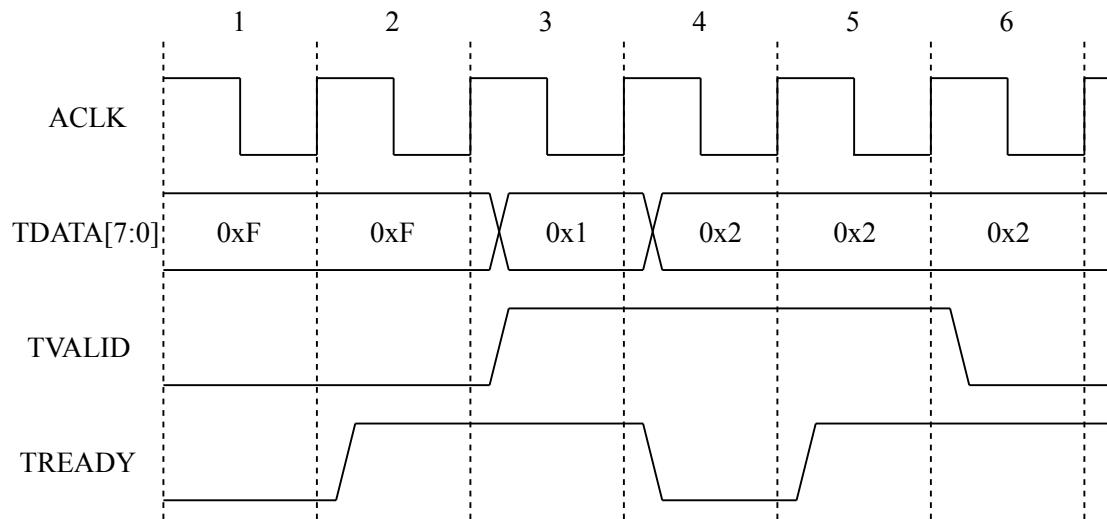
- TDATA - N-bajtowa równoległa szyna danych
- TVALID - sygnał informujący odbiorcę o obecność danych na szynie danych TDATA (1 - poprawne, 0 - brak)
- TREADY - sygnał informujący nadawcę o gotowości do przyjęcia danych przez odbiorcę (1 - gotowość, 0 - brak gotowości)

Na rysunku 2.13 przedstawiono kierunki generowanych sygnałów.



Rysunek 2.13: Schemat blokowy protokołu *AXI4-Stream*.

Na rysunku 2.14 przedstawiono przykładową transakcję przesyłu danych. Dane odczytywane są przy opadającym zboczu zegara. Transakcja przesyłu jest zatwierdzona, jeżeli sygnały TVALID i TREADY są w stanie wysokim, jak ma to miejsce w 3 i 5 cyklu.



Rysunek 2.14: Przykładowe transakcje z wykorzystaniem protokołu *AXI4-Stream*.

Protokół posiada także inne opcjonalne sygnały, które szczegółowo opisane są w dokumentacji [28].

2.6.5 Visual Studio Community

Visual Studio [13] to środowisko firmy Microsoft, które umożliwia tworzenie skryptów, bibliotek, programów konsolowych i okienkowych w językach C#, C++, Visual Basic i innych na systemy Windows, Mac, Android, iOS. Wspiera także tworzenie rozwiązań webowych i chmurowych. Posiada wiele funkcji wspomagających tworzenie i debugowanie kodu. Integruje system kontroli wersji GIT.

2.6.6 Python

Python [19] to wysokopoziomowy język z dużą liczbą standardowych bibliotek. Narzucona składnia w postaci odpowiedniego formatowania kodu poprawia czytelność. Bardzo dobrze nadaje się do tworzenia wszelkiego rodzaju testów. Brak potrzeby kompilacji skraca czas potrzebny do uruchomienia testu po jego modyfikacji.

Scapy

Do wstępniego testowania projektu podczas jego tworzenia wykorzystano bibliotekę Scapy [22]. Biblioteka pozwala na wysłanie dowolnej ramki z dowolnego portu sieciowego na komputerze PC.

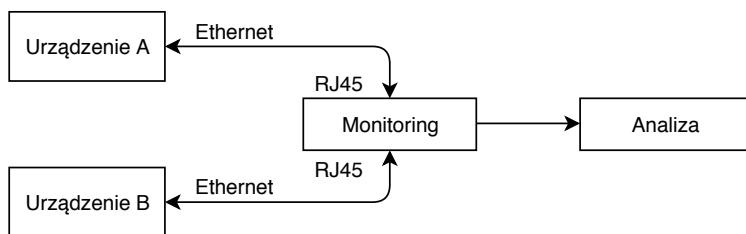
2.6.7 Trafgen

Trafgen to oprogramowanie dostępne na systemy Linux, które umożliwia generację bardzo dużego ruchu sieciowego na wybranym porcie komunikacyjnym. Program wchodzi w skład pakietu *Netsniff-NG* [15]. Program ze względu na swoją wielowątkową strukturę działania jest w stanie praktycznie nasycić łącze, nawet przy generacji ramek o minimalnej długości. Program pozwala definiować zawartość oraz rozmiar ramek ethernetowych, a także przypisywać generację ramek do poszczególnych wątków procesora. Przydatną opcją jest także możliwość generacji określonej liczby ramek, dzięki czemu oprogramowanie może sprawdzić się w trakcie testowania stabilności prezentowanego w pracy systemu poprzez sprawdzenie liczby utraconych pakietów.

Rozdział 3

Przedmiot pracy

Celem pracy było zaprojektowanie i zbadanie systemu pozwalającego na monitoring oraz analizę ruchu sieciowego. Ogólny schemat takiego systemu został pokazany na rysunku 3.1.

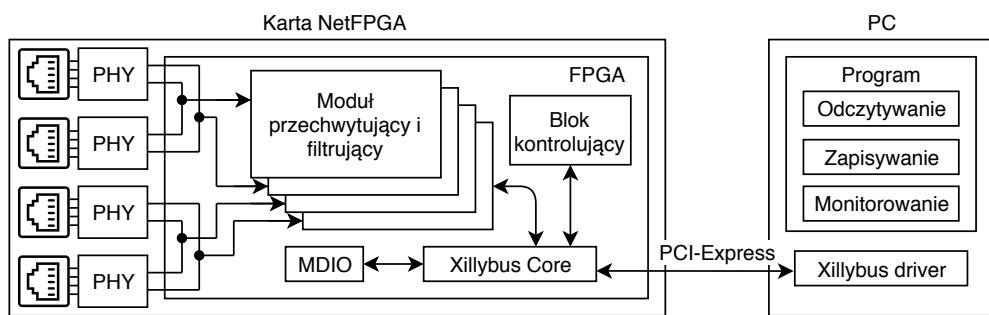


Rysunek 3.1: Schemat blokowy systemu monitorująco-analizującego.

Monitorowanie łącza ma na celu przechwycenie całego ruchu sieciowego, kolejno analiza takich danych jest potrzebna do wyciągnięcia potrzebnych informacji i parametrów.

3.1 Opis systemu

Prezentowany system składa się z dwóch elementów. Głównym komponentem jest karta deweloperska NetFPGA, która wpięta jest w slot PCI-Express komputera stacjonarnego. Na karcie wgrany jest projekt stworzony w środowisku Vivado. Na komputerze stacjonarnym zainstalowanym systemem operacyjnym Windows znajduje się program napisany w języku C++ do konfiguracji rejestrów oraz odbierania i zapisywania przetworzonych danych z karty NetFPGA. Komunikacja pomiędzy kartą, a komputerem zrealizowana jest wykorzystując połączenie PCI-Express. Na rysunku 3.2 został przedstawiony schemat blokowy systemu.



Rysunek 3.2: Schemat blokowy systemu.

Funkcje systemu:

- przechwytywanie ruchu sieciowego,
- filtracja dla zmniejszenia ilości danych,
- zmiana nastaw filtrów w czasie pracy,
- znaczniki czasowe z rozdzielcością 4ns,
- rozłączanie tunelu w wybranym kierunku,
- obsługa pakietów jumbo do 16382 bajtów,
- brak utraty pakietów przy saturacji łącząca sieciowego niezależnie od rozmiaru pakietu dla nasłuchiwanego dwóch portów.

3.1.1 Zasada działania

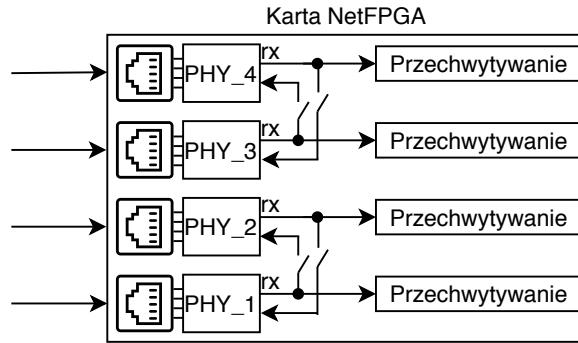
System posiada dwa tunele, dzięki którym możliwe jest nasłuchiwanie dwóch osobnych połączeń sieciowych w obu kierunkach. Karta odbiera ramkę ethernetową i przekazuje ją do przeciwnego układy PHY, który wysyła ramkę dalej, tworząc wspomniany tunel. Oprócz tego, odebrana ramka jest przekazywana dalej do kolejnej części układu. Następnie do ramki dodawany jest znacznik czasowy o rozdzielcości 4 ns oraz długość ramki. Format całego pakietu w tym momencie jest identyczny jak w pliku z rozszerzeniem *.pcap*, dzięki czemu z poziomu programu nie jest wymagane przetwarzanie strumienia pakietów, a wyłącznie odczytywanie i zapisywanie do pliku. Pakiet kolejno przechodzi przez blok filtrujący (dokładny opis w rozdziale 3.2.3), który posiada 4 filtry. Jeżeli nastawy przynajmniej 1 z 4 filtrów zostaną spełnione to dany pakiet zostaje przekazany do bloku Xillybus Core, który następnie z wykorzystaniem połączenia PCI-Express transmitsuje dane do systemu operacyjnego. Program, który jest uruchomiony na komputerze stacjonarnym w jednym wątku odbiera dane i zapisuje do bufora cyklicznego, a w drugim wątku odczytuje z bufora cyklicznego i zapisuje na dysk twardy do pliku o rozszerzeniu *.pcap*.

3.1.2 Możliwe konfiguracje

System może działać w różnych konfiguracjach. Głównym założeniem systemu jest tworzenie tunelu i monitorowanie ruchu sieciowego danego połączenia, ale system ma także inne możliwości.

Przechwytywanie

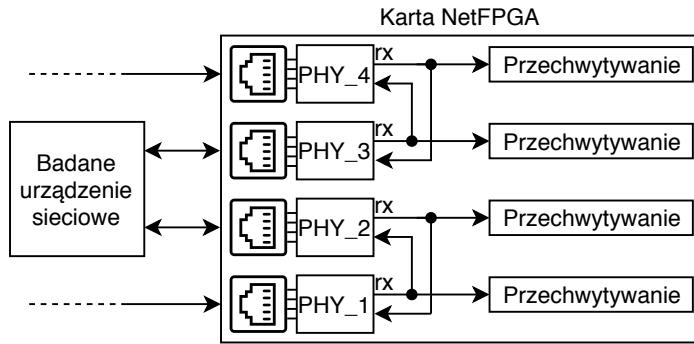
System po wyłączeniu tuneli jest w stanie odbierać przychodzące dane w standardach 100/1000BASE-T. Schemat konfiguracji został przedstawiony na rysunku 3.3.



Rysunek 3.3: Konfiguracja karty NetFPGA z wyłączeniami tunelami.

Pomiar czasu

Dwa tunele umożliwiają precyzyjny pomiar opóźnień wprowadzanych przez badane urządzenie np. przełącznik sieciowy, router albo sterownik przemysłowy. Dzięki precyzyjnym znacznikom czasowym w dwóch miejscach, możliwe jest obliczenie czasu przejścia ramki przez urządzenie. Omawiana konfiguracja została przedstawiona na rysunku 3.4.

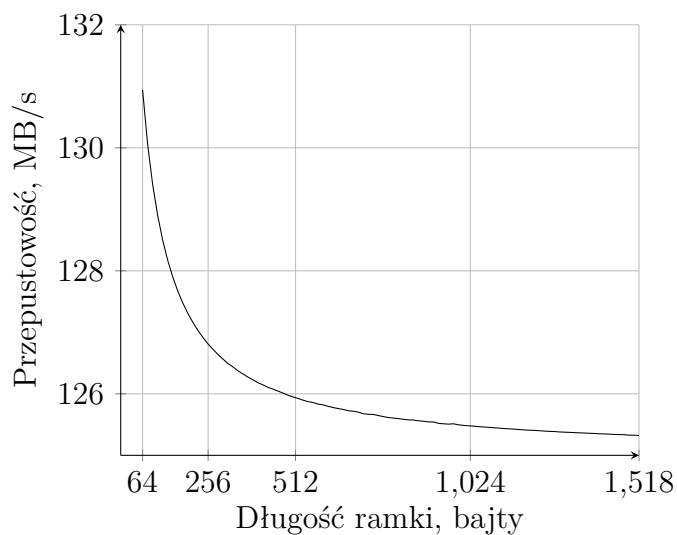


Rysunek 3.4: Konfiguracja karty NetFPGA do pomiaru opóźnień wprowadzanych przez badane urządzenie.

3.1.3 Ograniczenia teoretyczne

Pierwotnie projekt miał przechwytywać ruch sieciowy przechodzący przez wyłącznie jedno połączenie ethernetowe, jednak warunki okazały się na tyle dobre, że udało się zduplikować rozwiązanie. Ostatecznie system umożliwia przechwytywanie

dwóch połączeń. Jedynym ograniczeniem systemu jest przepustowość połączenia PCI-Express, która w przypadku zastosowania podstawowej wersji bloku Xillybus wynosi około 400 MB/s. Pojedynczy strumień danych (których łącznie jest 4) jest w stanie wygenerować przy maksymalnym natężeniu ruchu sieciowego do 131 MB/s danych (64-bajtowa pełna ramka Ethernet + 8-bajtowa preambuła + 12-bajtowy odstęp między ramkami). Zwiększoną ilość danych wynika z dodatkowego narzutu na rozmiar przekazywanej ramki, która poszerzona jest o pola znacznika czasowego i rozmiaru ramki (dodatkowe 16 bajtów). Zależność została przedstawiona na rysunku 3.5.



Rysunek 3.5: Zależność między długością ramki ethernetowej, a wymaganą przepustowością połączenia PCI-Express przy pełnym nasyceniu łącza.

3.1.4 Plan prac

Projekt został wykonany metodą *bottom-up* tzn. pracę rozpoczęto od podstawowych funkcji potrzebnych do realizacji kolejnych. Większość tworzonych bloków została najpierw wstępnie symulowana na tzw. testbenchu, kolejno syntezowana i implementowana. Podczas każdego z etapów mogą pojawiać się błędy. Dzieje się tak, ponieważ język Verilog nie zapewnia syntezы każdego zgodnego z standardem języka kodu źródłowego. Istnieją pewne ograniczenia, o których należy pamiętać

przy projektowaniu modułów. Dodatkowo każdy blok musi spełniać odpowiednie ramy czasowe narzucone przez częstotliwość taktowania danego modułu.

Praca została podzielone na etapy, po których za każdym razem sprawdzane było poprawne działanie poprzednio zrealizowanych elementów:

1. Sprawdzenie działania karty NetFGPA poprzez wgranie prostego projektu.
2. Analiza dostępnych rozwiązań PCI-Express.
3. Implementacja interfejsu Xillybus obsługującego połączenie PCI-Express.
4. Stworzenie programu odbierającego i wysyłającego dane po połączeniu PCI-Express.
5. Stworzenie bloku odbierającego oraz nadającego ramki Ethernet z obsługą interfejsu RGMII w trybie 1000 Mbit/s.
6. Stworzenie tunelu pomiędzy dwoma układami PHY.
7. Przekazywanie wszystkich danych przesyłanych pomiędzy układami PHY do użytkownika.
8. Stworzenie bloku pakietyzującego odebrane ramki, który umożliwia rozróżnienie ramek w ciągłym strumieniu danych.
9. Stworzenie bloku filtrującego spakietyzowane ramki.
10. Stworzenie bloku kontrolnego, który zapewnia kontrolę przepływu danych do użytkownika.
11. Dodanie obsługi prędkości 100 Mbit/s.
12. Dodanie obsługi interfejsu MDIO.

Podczas tworzenia projektu sprzętowego równolegle rozwijano program do którego dodawane były kolejne funkcje. Dokładny opis programu znajduje się w rozdziale 3.3.

Początkowo pominięta została implementacja interfejsu MDIO ze względu na brak potrzeby ingerencji w rejesty układow PHY. Pierwsze badania układu pomiarowego wykazały pewną anomalię w aspekcie wprowadzanych przez układ

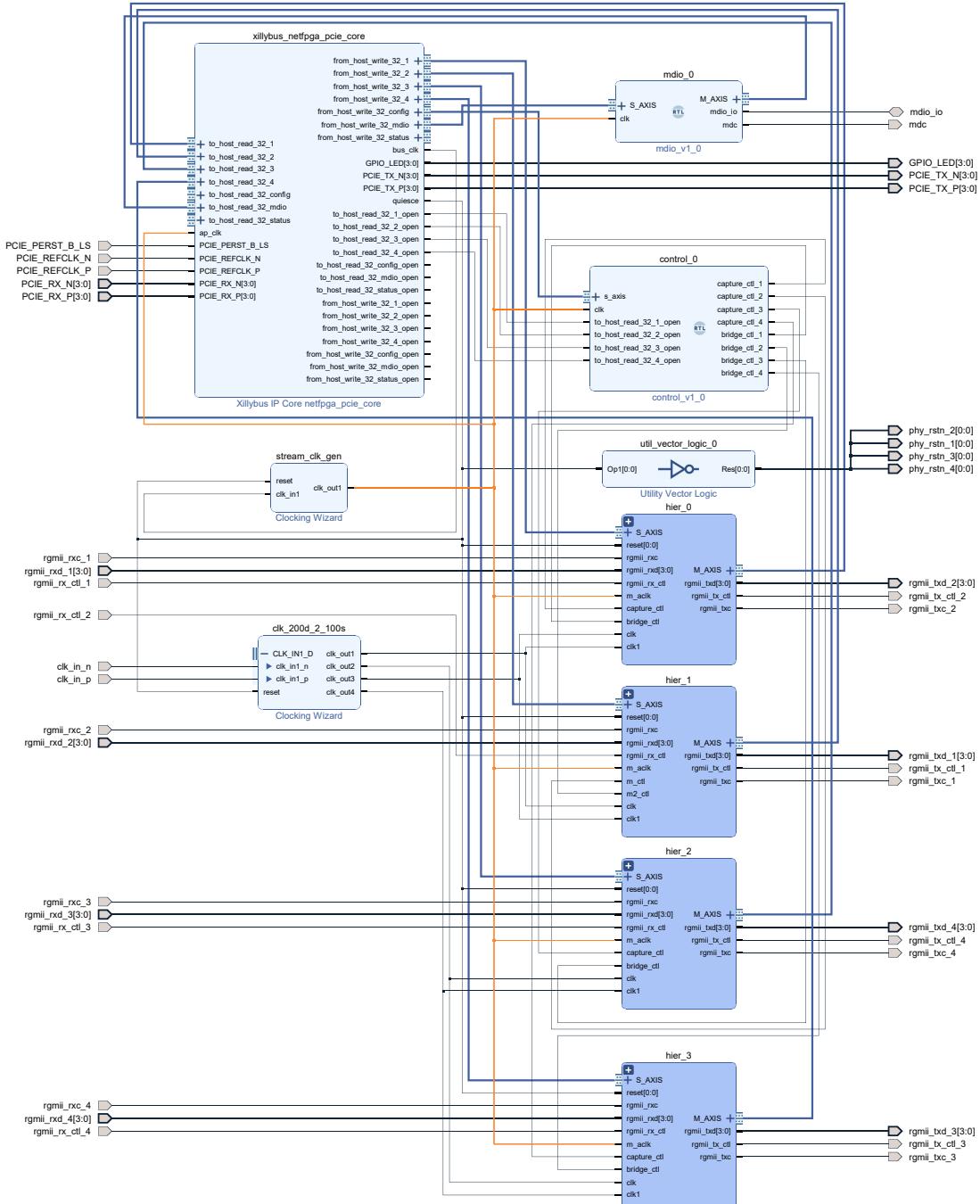
opóżnień, które były dużo wyższe niż przewidywane. Problemem okazała się załączona domyślnie opcja oszczędzania energii na układach PHY. Problem został opisany szerzej w rozdziale 3.4.2

3.2 Część sprzętowa

Projekt części sprzętowej, który znajduje się na układzie FPGA składa się z czterech identycznych modułów oraz kilku dodatkowych bloków odpowiadających za komunikację oraz kontrolę układów PHY. Zrzut ekranu całego projektu z programu Vivado znajduje się na rysunku 3.6.

Projekt sprzętowy znajdujący się na karcie odpowiedzialny jest za:

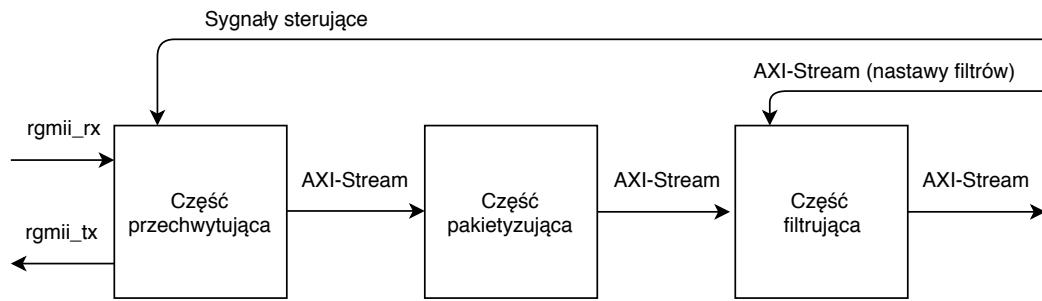
- Odbieranie ramek z układu PHY i przekazanie jej do sąsiadującego układu PHY (tworząc tunel z sąsiadującym układem) oraz przekazywanie ramek do dalszej części projektu - część przechwytyująca (rozdział 3.2.1).
- Pakietyzację ramek do odpowiedniego formatu, czyli dodanie znacznika czasowego i liczby bajtów ramki - część pakietyzująca (rozdział 3.2.2).
- Filtrację pakietów według odebranych nastaw - część filtrująca (rozdział 3.2.3).
- Kontrolę rejestrów układów PHY - blok MDIO (rozdział 3.2.4).
- Kontrolę przepływu danych - blok kontrolny (rozdział 3.2.5).
- Wysyłanie i odbieranie danych do i od użytkownika - wymiana danych (rozdział 3.2.6).



Rysunek 3.6: Zrzut ekranu z blokowego schematu projektu sprzętowego w programie Vivado.

W całym projekcie przekazywanie danych pomiędzy poszczególnymi blokami odbywa się z wykorzystaniem pojedynczych sygnałów asynchronicznych lub synchronicznych oraz interfejsu *AXI4-Stream*, który opisany został w rozdziale 2.6.4.

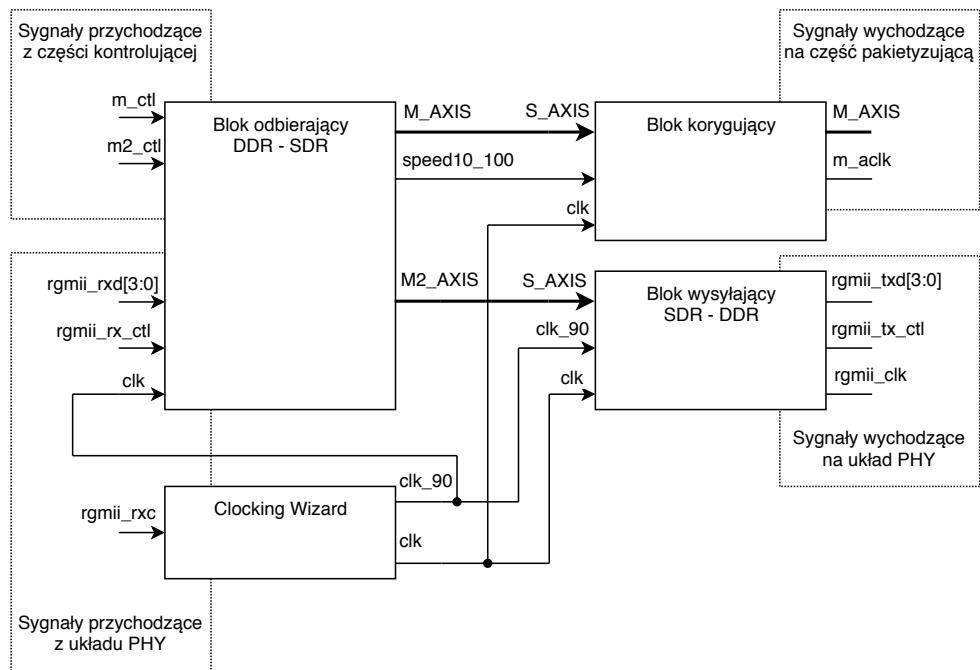
Wspomniany na początku rozdziału moduł składa się z 15 bloków. Moduł ze względu na duże skomplikowanie został podzielony na kilka części, które zostaną przedstawione i opisane w kolejnych podrozdziałach. Uogólniony schemat blokowy pojedynczego modułu znajduje się na rysunku 3.7.



Rysunek 3.7: Uproszczony schemat blokowy pojedynczego modułu.

3.2.1 Przechwytywanie

Część przechwytyująca składa się z 4 bloków, której schemat blokowy jest przedstawiony na rysunku 3.8. Wszystkie bloki w tej części pracują synchronicznie do sygnału zegarowego RXC interfejsu RGMII (opis interfejsu w rozdziale 2.1.2).



Rysunek 3.8: Schemat blokowy części przechwytywającej.

Blok odbierający

Pierwszym z nich jest blok odbierający dane z układu PHY, który konwertuje sygnał RGMII do formatu *AXI4-Stream* bez kontroli przepływu o szerokości szyny 8-bitów. Odkodowuje prędkość połączenia, którą można odczytać na liniach sygnałowych interfejsu podczas przerwy pomiędzy kolejnymi ramkami (dokładny opis w rozdziale 2.1.2). Blok posiada także dwa asynchroniczne sygnały wejściowe, które kontrolują odpowiednio dwa wyjścia *AXI4-Stream*. Kontrola polega na możliwości zdalnego włączenia/wyłączenia danego strumienia. Implementacja mechanizmu zezwala na zmianę stanu wyłącznie podczas braku transmisji, czyli w momencie przerwy między ramkami, dzięki czemu nie ma możliwości uszkodzenia wysyłanej w danym momencie ramki. Sygnały kontrolujące generowane są przez część kontrolującą układu, której opis znajduje się rozdziale 3.2.5.

Blok Clocking Wizard

Blok *Clocking Wizard* jest wykorzystany do przesunięcia fazowego o 90 stopni przychodzącego zegara z interfejsu RGMII oraz kondycjonowanie samego sygnału zegarowego. Taki zabieg jest potrzebny ze względu na specyfikę interfejsu RGMII.

Blok korygujący

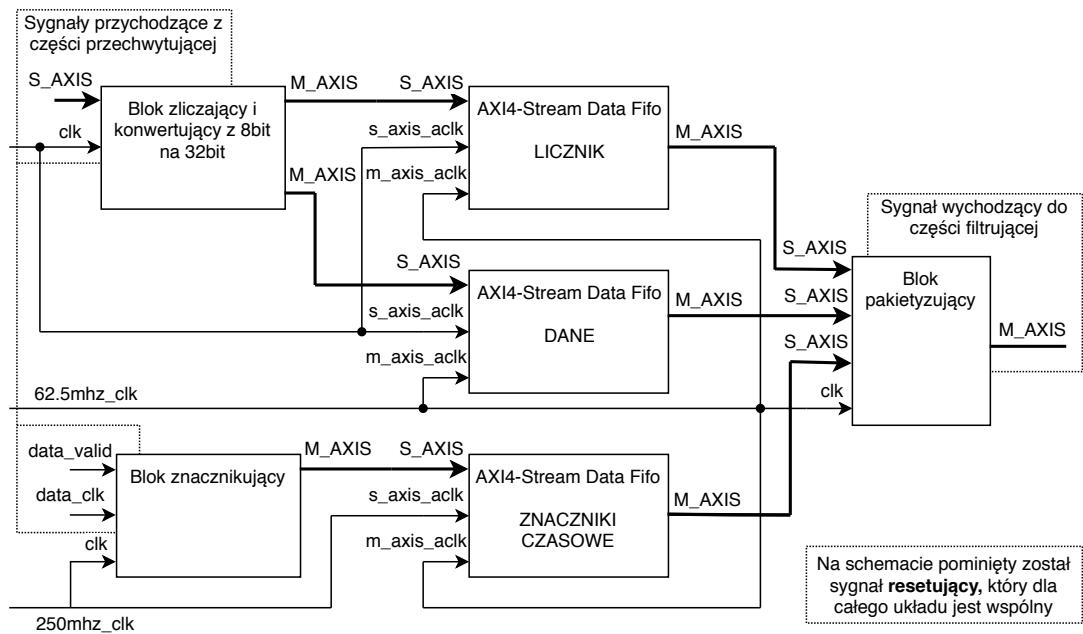
Blok korygujący służy do korekcji danych strumienia oraz generacji sygnału zegarowego, jeżeli prędkość linku wynosi 100 Mbit/s. Taki mechanizm jest potrzebny ze względu na sposób działania interfejsu RGMII, który w przypadku niższych prędkości niż 1000 Mbit/s, generuje dane w trybie SDR (ang. Single Data Rate) zamiast DDR (ang. Double Data Rate). Zmiana sposobu przesyłania danych powoduje, że odebrane dane w trybie 100 Mbit/s są podwojone tzn. każdy bajt wychodzący z bloku odbierającego posiada zduplikowane półabajty (np. 0xCC, 0xDD zamiast 0xCD). Z tego względu w przypadku niższych prędkości, blok generuje także sygnał zegarowy z o połowę niższą częstotliwością.

Blok wysyłający

Blok wysyłający jest odpowiedzialny za ponowną konwersję sygnału z postaci *AXI4-Stream* do RGMII.

3.2.2 Pakietyzacja ramek

Część pakietyzująca składa się z 6 bloków z czego 3 to bloki *AXI4-Stream Data Fifo* (opis w rozdziale 2.6.3). Głównym założeniem tej części jest przetworzenie przychodzących ramek i dodanie nagłówka, który upodabnia strukturę strumienia danych do struktury pliku z rozszerzeniem *.pcap* (opis w rozdziale 2.1). Dodatkowym zadaniem bloku jest synchronizacja zegarów i konwersja z 8-bitowej szyny danych do 32-bitowej szyny danych. Schemat blokowy tej części został przedstawiony na rysunku 3.9.



Rysunek 3.9: Schemat blokowy części pakietyzującej.

Bloki AXI4-Stream Data Fifo

W tej części projektu wykorzystywane są 3 bloki buforujące FIFO. Mają za zadanie nie tylko przechowywanie tymczasowe ramek i wartości, ale także synchronizację pomiędzy różnymi sygnałami zegarowymi. Bloki zostały skonfigurowane w następujący sposób:

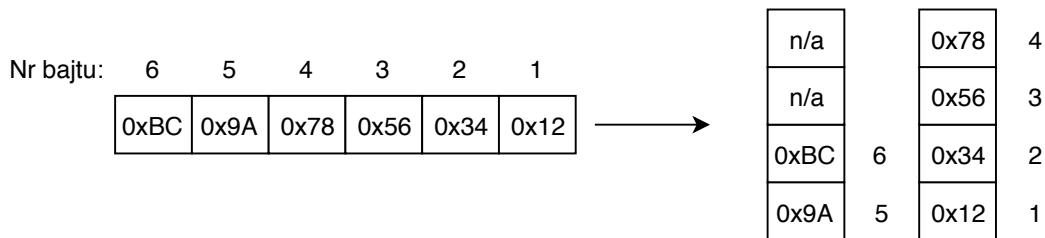
- FIFO_LICZNIK - 2-bajtowa szerokość szyny danych, głębokość bufora 1024;
- FIFO_DANE - 4-bajtowa szerokość szyny danych, głębokość bufora 4096;
- FIFO_ZNACZNIKI_CZASOWE - 4-bajtowa szerokość szyny danych, głębokość bufora 1024;

Buforowanie danych jest wymagane ze względu na brak kontroli przepływu odbieranych danych z części przechwytyjącej.

Blok zliczający i konwertujący

Dane przychodzące z części przechwytyjącej (rozdział 3.2.1) są zliczane przez 14-bitowy licznik oraz konwertowane z 8-bitowej szyny danych na 32-bitową szynę

danych w sposób pokazany na rysunku 3.10. 4-bajtowe części ramki są przekazywane do bufora FIFO_DANE. Po przetworzeniu całej ramki wartość licznika jest wysyłana do bufora FIFO_LICZNIK. Moment ten inicjuje rozpoczęcie działania bloku pakietyzującego.



Rysunek 3.10: Schemat przedstawiający konwersję 8-bitowej szyny do 32-bitowej szyny.

Blok znacznikujący

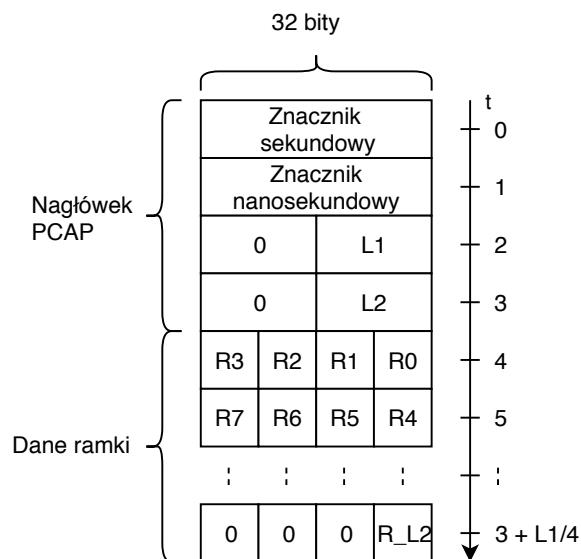
Blok znacznikujący posiada zaimplementowane dwa liczniki. Pierwszy 28-bitowy licznik zlicza cykle sygnału zegarowego o częstotliwości 250 MHz. W momencie, kiedy licznik ma przekroczyć wartość 250mln to jest resetowany, a drugi 32-bitowy licznik inkrementowany. W ten sposób blok posiada zegar o rozdzielczości 4 ns składający się z dwóch części: sekundowej oraz nanosekundowej.

Moment pojawienia się ramki na bloku odbierającym (rozdział 3.2.1) jest sygnałem, który inicjuje odczytanie wartości liczników. Wartości kolejno są wysyłane do bufora FIFO_ZNACZNIKI_CZASOWE w kolejności: licznik sekundowy, licznik nanosekundowy. Należy zwrócić uwagę, że przy wysyłaniu licznika nanosekundowego wartość przesuwana jest o 2 bity w lewo (operacją równoważną jest pomnożenie przez 4), aby uzyskać prawdziwą wartość nanosekund. Taki format danych został narzucony przez strukturę nagłówka ramki pliku *PCAP*.

Blok pakietyzujący

Blok pakietyzujący rozpoczyna swoją pracę w chwili pojawienia się danych w buforze FIFO_LICZNIK oraz FIFO_ZNACZNIKI_CZASOWE. Moment pojawienia się tych danych oznacza, że we wszystkich 3 buforach znajdują się dane potrzebne do stworzenia pakietu. Blok zaczyna od przekazania na wyjście wartości

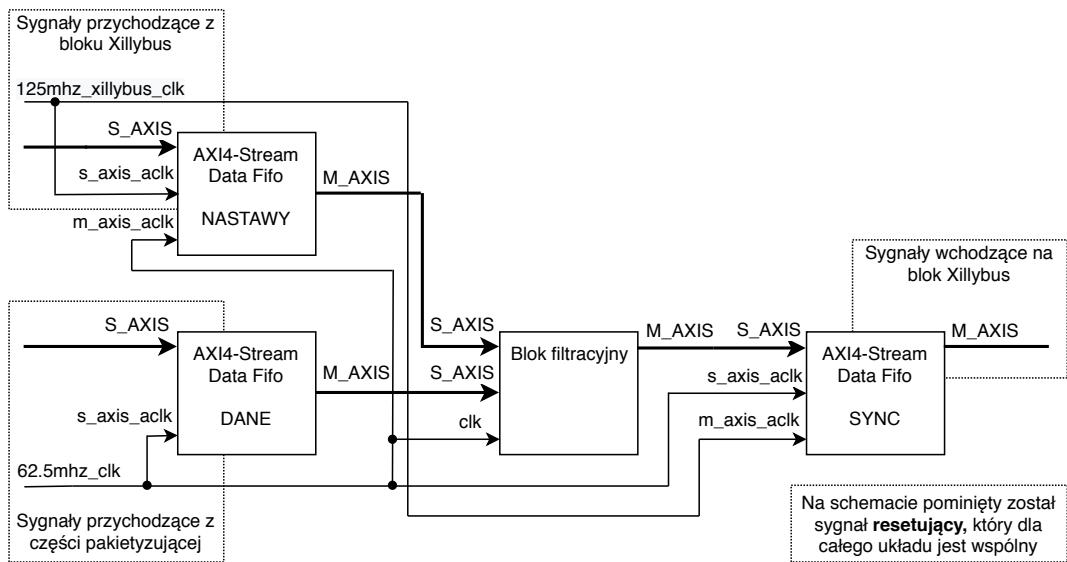
sekundowej i nanosekundowej z bufora FIFO_ZNACZNIKI_CZASOWE. Kolejno z bufora FIFO_LICZNIK blok odbiera długość ramki i wysyła na wyjściu dwie wartości: długość ramki zaokrągloną w górę do wartości podzielnej przez 4 oraz długość ramki bez zmian. Następnie zaczyna odczytywać dane ramki z bufora FIFO_DANE i wysyłać na wyjście. Struktura wyjściowa strumienia danych została przedstawiona na rysunku 3.11.



Rysunek 3.11: Struktura wyjściowa strumienia danych.

3.2.3 Filtracja

Część filtrująca składa się z 4 bloków: 3 bloków buforujących *AXI4-Stream Data Fifo* oraz bloku filtracyjnego. Blok filtracyjny jest najbardziej złożonym blokiem w całym projekcie. Główną funkcją tej części to przepuszczanie pakietów spełniających ustawione reguły oraz synchronizacja z blokiem *Xillybus*. Blok filtracyjny posiada łącznie 4 filtry, każdy z 28 polami konfiguracyjnymi oraz jednym polem określającym minimalną długość pakietu. Schemat blokowy tej części został przedstawiony na rysunku 3.12.



Rysunek 3.12: Schemat blokowy części filtracyjnej.

Bloki AXI4-Stream Data Fifo

W tej części projektu wykorzystywane są 3 bloki buforujące FIFO. Mają za zadanie nie tylko przechowywanie tymczasowe ramek i wartości, ale także synchronizację pomiędzy różnymi sygnałami zegarowymi. Bloki zostały skonfigurowane w następujący sposób:

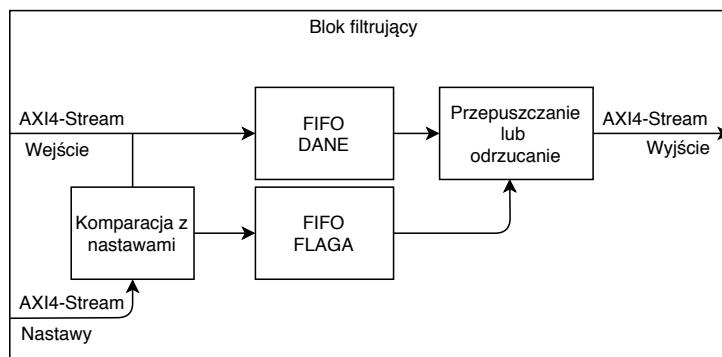
- FIFO_NASTAWY - 4-bajtowa szerokość szyny danych, głębokość bufora 512;
- FIFO_DANE - 4-bajtowa szerokość szyny danych, głębokość bufora 4096;
- FIFO_SYNC - 4-bajtowa szerokość szyny danych, głębokość bufora 1024;

Bufor FIFO_SYNC jest elementem synchronizującym pomiędzy częścią filtracyjną, która działa z częstotliwością 62.5 MHz, a blokiem Xillybus, który działa z sygnałem zegarowym o częstotliwości 125 MHz. Podobną funkcję spełnia bufor FIFO_NASTAWY, który konwertuje sygnał *AXI4-Stream* o częstotliwości 125 MHz na 62.5 MHz. Bufor FIFO_DANE jest potrzebny, ponieważ w momencie odbierania nowej konfiguracji nastaw, filtracja musi zostać zatrzymana. Do-

datkowo jest to spowodowane tym, że blok pakietyzujący nie posiada wejścia na sygnał kontroli przepływu (brak sygnału TREADY).

Blok filtracyjny

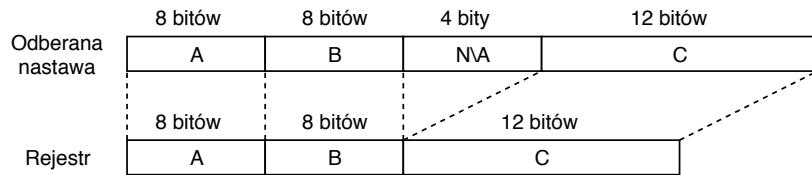
Blok filtracyjny jest głównym elementem filtracji. Schemat bloku filtracyjnego jest przedstawiony na rysunku 3.13. Blok odbiera ramkę oraz porównuje co cykl z nastawami, według zaprojektowanego algorytmu. Porównywanie odbywa się w sposób potokowy. Odebrana ramka jest bezpośrednio przekazywana do tymczasowego bufora FIFO_DANE w trakcie jej przetwarzania. Po przetworzeniu całej ramki do bufora FIFO_FLAGA jest wysyłana flaga z informacją, że ramkę należy przepuścić lub odrzucić oraz informacja o długości ramki. Odrzucanie polega na odczytaniu danych z bufora FIFO_DANE z ustawioną błędnią flagą poprawności danych na wyjściu (sygnał TVALID w stanie niskim). Taki mechanizm gwarantuje możliwość przetwarzania kolejnej ramki w trakcie odrzucania/przepuszczania.



Rysunek 3.13: Schemat bloku filtracyjnego.

Odbieranie nastaw ma zawsze wyższy priorytet od przetwarzania ramki. W chwili pojawienia się sygnału informującego o nowych nastawach (sygnał TVALID z bloku FIFO_NASTAWY na rysunku 3.12), blok czeka na zakończenie przetwarzania bieżącej ramki. Po zakończeniu przetwarzania filtracja jest zatrzymywana, a nastawy do wszystkich 4-filtrów są odbierane i zapisywane do rejestrów konfiguracyjnych. Nastawy są odbierane jako jeden pakiet składający się z 128 elementów po 4 bajty. Jako, że odbierane nastawy posiadają inną szerokość szyny (32 bity) od

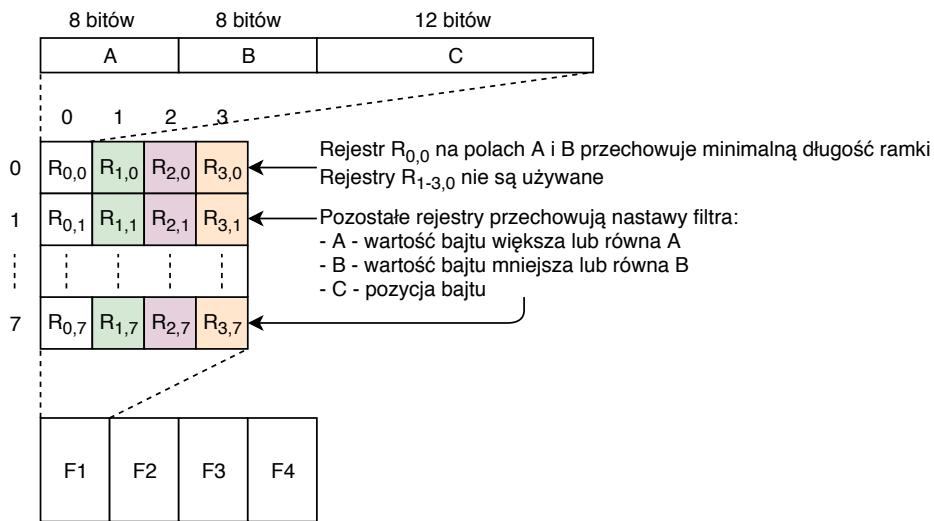
szerokości komórki rejestru (28 bitów), nastawy są rzutowane w sposób pokazany na rysunku 3.14.



Rysunek 3.14: Rzutowanie odbieranej nastawy na komórkę rejestru konfiguracyjnego.

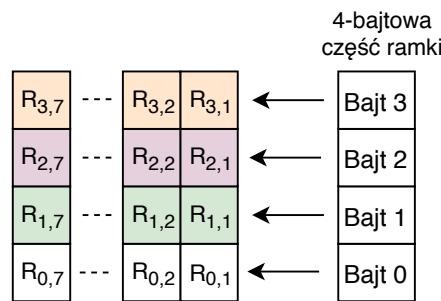
Struktura rejestrów konfiguracyjnych

Struktura pamięci przechowującej nastawy filtrów została przedstawiona na rysunku 3.15. Każdy z 4 filtrów (F1 - F4) dostępnych w bloku filtrującym posiada 32 rejesty. Każdy register ma długość 28 bitów. $R_{0,0}$ przechowuje minimalną długość ramki na polach A i B. Rejestry $R_{1-3,0}$ nie są używane. Pozostałe rejesty przechowują nastawy danego filtra. Można powiedzieć, że przedstawiona struktura jest 3-wymiarową tablicą o rozmiarze 4 (4 kolumny nastaw) na 8 (8 wierszy nastaw) na 4 (4 filtry).



Rysunek 3.15: Struktura rejestrów konfiguracyjnych

W każdym cyklu zegarowym do 4-bajtowej części ramki porównywane są po 4 nastawy z każdego z 4 filtrów (łączenia 16 nastaw w jednym cyklu). W pojedynczym filtrze porównywanie odbywa się według określonej struktury. Pozycja nastawy (pierwszy indeks $R_{0-3, _}$) w rejestrze konfiguracyjnym determinuje pozycję porównywanego bajtu w danym cyklu. Oznacza to, że jedna z nastaw z indeksem $R_{0, _}$ jest porównywana zawsze do bajtu z indeksem 0, nastawa z indeksem $R_{1, _}$ jest porównywana do bajtu z indeksem 1 itd., co zostało pokazane na rysunku 3.16. Każdy bajt z 4-bajtowej części ramki jest filtrowany niezależnie od pozostałych, jednakże w momencie niespełnienia jednej z nastaw ustawiana jest w danym filtrze flaga Skip_packet.



Rysunek 3.16: Przedstawienie zależności pozycji bajtu w 4-bajtowej części ramki do nastaw.

Pola nastaw

Pole nastaw filtra składa z trzech wartości:

- A - 8-bitowa wartość określająca, że porównywana wartość bajtu ma być większa lub równa.
- B - 8-bitowa wartość określająca, że porównywana wartość bajtu ma być mniejsza lub równa.
- C - 12-bitowa wartość określająca numer cyklu.

Numer cyklu razem z położeniem nastawy w rejestrze konfiguracyjnym określa pozycję porównywanego bajtu. Na przykład, jeżeli wartość C wynosi 2, a nastawa znajduje się w rejestrze $R_{0, _}$ to sprawdzana będzie wartość 8 bajtu (licząc od 0) w

ramce ethernetowej. Jeżeli wartość C wynosi 3, a nastawa znajduje się w rejestrze $R_{3, _}$ to sprawdzana będzie wartość 15 bajtu (licząc od 0) w ramce. Wzór na pozycję bajtu w ramce można zdefiniować jako:

$$P = 4 \cdot C + N \quad (3.1)$$

Gdzie,

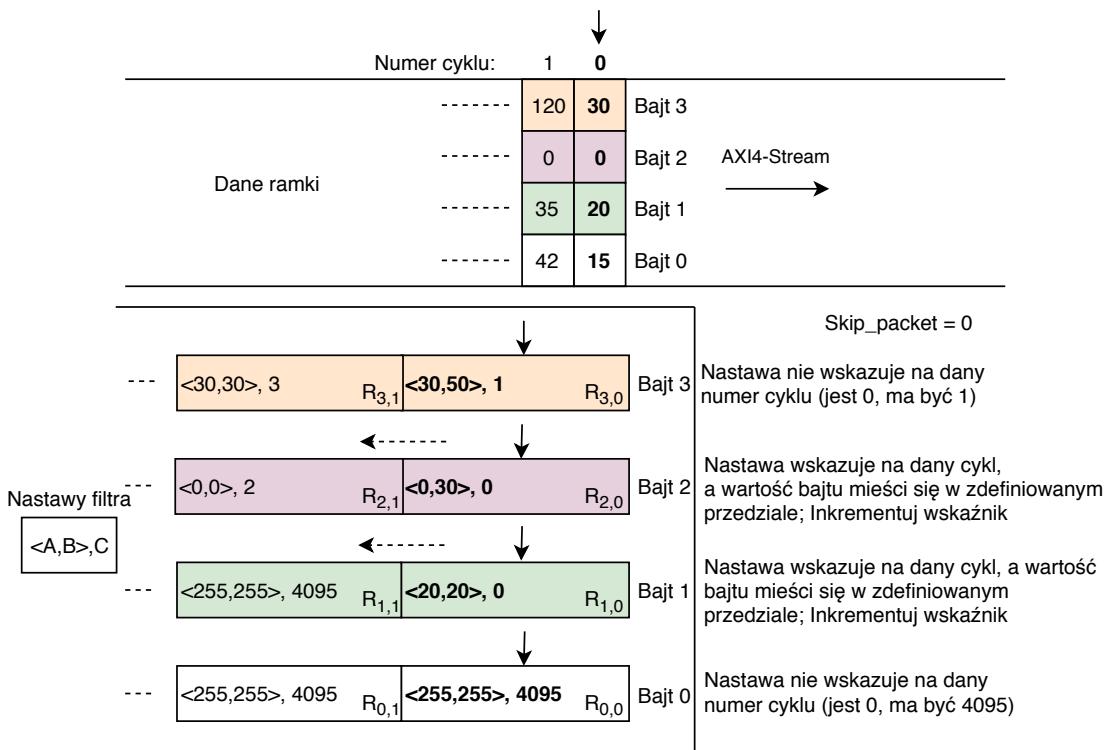
- P - pozycja w ramce ethernetowej,
- N - pierwszy indeks rejestru $R_{N, _}$,
- C - wartość numeru cyklu w nastawie.

Nastawa zostaje niespełniona, kiedy wartość bajtu na określonej pozycji nie mieści się w przedziale domkniętym $\langle A, B \rangle$.

Przykład działania filtracji

Na rysunku 3.17 i 3.18 przedstawiono 2 cykle wizualizujące mechanizm działania filtracji dla pojedynczego filtra. Pozostałe 3 filtry w danym bloku filtracyjnym działają równolegle w ten sam sposób i porównują jednocześnie te same wartości ramki. W przykładzie został pominięty początkowy etap przetwarzania nagłówka Rysunek 3.17 przedstawia pierwszy cykl filtracji. Wszystkie pomocnicze wskaźniki (oznaczone na schemacie strzałkami) nastaw są wyzerowane i wskazują na pierwsze elementy tablicy. Na rysunku 3.17 analizowana jest pierwsza 4-bajtowa część ramki składająca się z wartości [30, 0, 20, 15]. Proces ten można podzielić na 4 równoległe i niezależne części zależne od indeksu bajtu (3, 2, 1, 0). Wartości [30, 0, 20, 15] porównywane są kolejno z nastawami na które wskazuje strzałka.

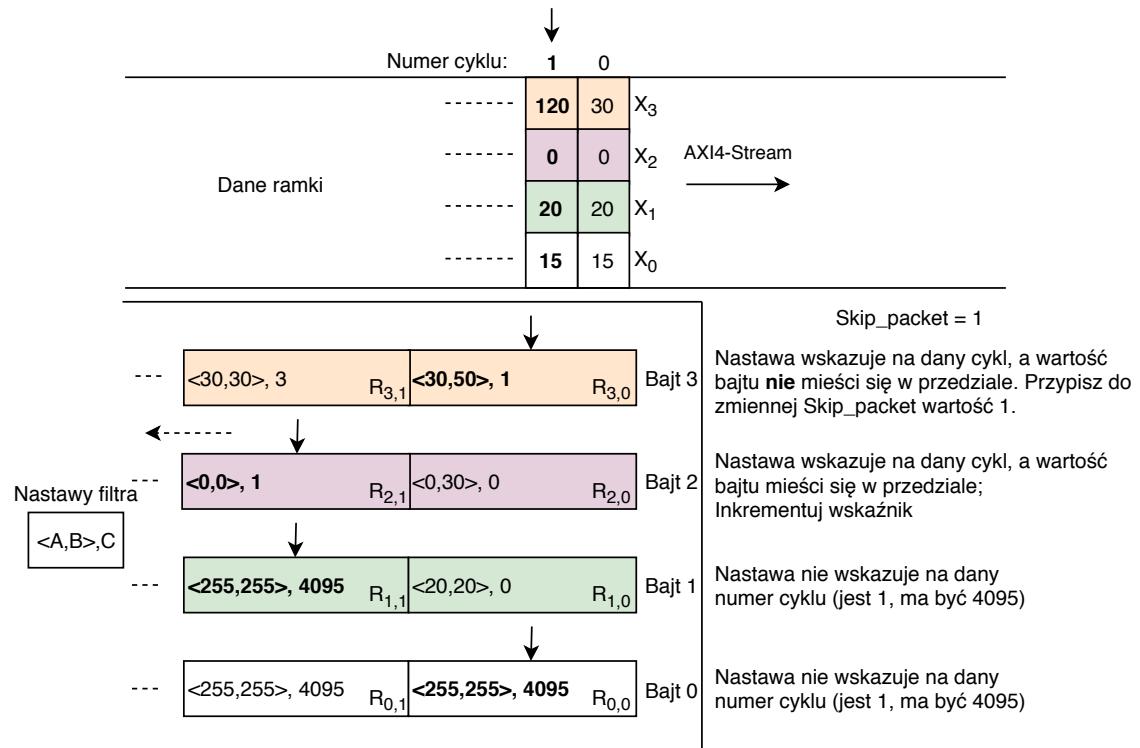
Na rysunku 3.17 wartość 30 jest porównywana z nastawą o indeksie $R_{3,0}$. Zauważać można, że dana nastawa nie wskazuje na aktualny numer cyklu (pozycję bajtu w ramce), dlatego porównywanie jest pomijane. Kolejna liczba 0 jest porównywana z nastawą o indeksie $R_{2,0}$. Nastawa wskazuje na aktualny numer cyklu oraz liczba mieści się w przedziale $\langle 0, 30 \rangle$. Oznacza to, że porównanie zostało spełnione, a ramka do tej pory spełnia nastawy filtra. Wskaźnik nastawy jest przesuwany na rejestr o indeksie $R_{2,1}$. Analogiczny proces zachodzi w kolejnych dwóch wierszach.



Rysunek 3.17: Pierwszy cykl działania algorytmu filtracji

Rysunek 3.18 przedstawia przetwarzanie kolejnego 4-bajtowego segmentu ramki. Tym razem, pozycje wskaźników nastaw zostały już przesunięte analogicznie do działań wykonanych na rysunku 3.17. Można zauważyć, że wartość 120 z rejestru $R_{3,0}$ nie zgadza się z nastawą, gdzie warunkiem spełnienia nastawy jest wartość w przedziale $\langle 30, 50 \rangle$. Oznacza to, że przetwarzana ramka nie spełnia nastaw filtra, a co za tym idzie ustawiana w stan wysoki jest flaga Skip_packet. Przetwarzanie ramki odbywa się dalej, jednak nie ma to żadnego dalszego skutku.

Po przetworzeniu całej ramki następuje sprawdzenie flag skip_packet z wszystkich 4 filtrów. Jeżeli chociaż jeden z filtrów zakończył pracę z flagą Skip_packet ustawioną w stan niski, to wysyłana jest flaga do bufora FIFO_FLAG (rysunek 3.13) o przepuszczeniu ramki, a w innym przypadku o jego odrzucenie.



Rysunek 3.18: Drugi cykl działania algorytmu filtracji

Ograniczenia i wymagania bloku filtracyjnego

Potokowe przetwarzanie i porównywanie z nastawami filtrów wymagało pewnych zabiegów optymalizacyjnych. Jak można zauważyć na przykładzie opisany w poprzednim podrozdziale, nastawy są w pewien sposób uporządkowane. Każde 7 z 28 pól jest porównywane tylko z 1 z 4 bajtów w danym cyklu. Oznacza to, że występuje pewne ograniczenie, które powoduje, że maksymalna liczba możliwych do ustawienia pól, co prawda wynosi łącznie 28, jednakże pozycji bajtów spełniających zależność $N \bmod 4 = M$, gdzie N to pozycja bajtu w ramce, a M to numer porównywanego bajtu w cyklu, może być już maksymalnie 7. Kolejnym ograniczeniem jest brak możliwości porównania dwóch tych samych pozycji w ramce w pojedynczym filtrze. Wymaganiem co do odbieranych przez blok filtrujący nastaw, jest ich wcześniejsze posortowanie po pozycji. W innym przypadku filtracja nie bę-

dzie działać zgodnie z założeniami, ponieważ mechanizm filtracji zakłada, że każde następne pole wskazuje na wyższą od poprzedniego pola pozycję.

3.2.4 Blok MDIO

Blok MDIO implementuje obsługę interfejsu MDIO, która została opisana w rozdziale 2.1.3. Na rysunku 3.19 przedstawiono wygląd bloku kontrolnego z programu Vivado. Blok odbiera 32-bitową wartość wprowadzoną z programu przez użytkownika oraz konwertuje na interfejs MDIO. Kolejno po wykonaniu operacji zapisu lub odczytu rejestru PHY zostaje w takim samym formacie wysyłana z powrotem do użytkownika. Taki mechanizm zapewnia możliwość sprawdzenia poprawności wysłanych danych, a zarazem implementuje możliwość odczytu rejestrów.



Rysunek 3.19: Blok implementujący obsługę MDIO.

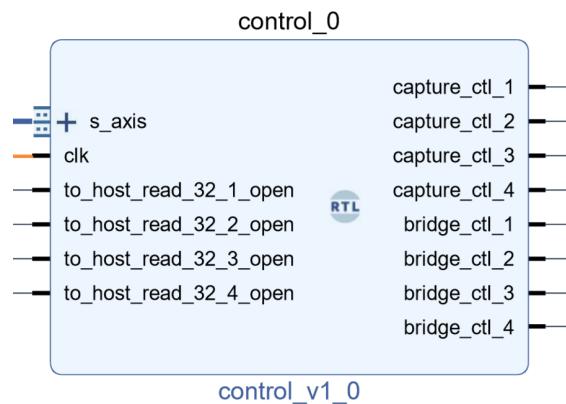
3.2.5 Blok kontrolujący

Blok kontrolujący odpowiada za włączanie/wyłączanie poszczególnych tuneli oraz przechwytywania w zależności od wartości rejestru. Sygnały parami wchodzą do bloków odbierających znajdujących się w części przechwytyjącej (rozdział 3.2.1) każdego z 4 głównych modułów. Blok posiada 8-bitowy rejestr, który można kontrolować poprzez wysłanie do bloku odpowiedniej wartości. Dane do bloku wysyłane są z programu (3.3) i odbierane przez blok Xillybus (3.2.6). W bloku została zaimplementowana możliwość załączania i wyłączania pojedynczych bitów bez ingerencji w pozostałe wartości. Osiągnięto to przez zarezerwowanie najstarszego bitu, który określa czy operacja jest ustawiająca lub kasująca bity. Przykład przedstawiający działanie mechanizmu został pokazany na rysunku 3.20.



Rysunek 3.20: Przykład przedstawiający działanie mechanizmu manipulacji wartościami rejestru.

Bity 0:3 kontrolują odpowiednio wyjściemi capture_ctl_1-4, a bity 4:7 kontrolują wyjściymi bridge_ctl_1-4. Na rysunku 3.21 przedstawiono wygląd bloku kontrolnego z programu Vivado. Można zauważyć, że do bloku oprócz interfejsu komunikacyjnego *AXI4-Stream* wchodzą sygnały sygnalizujące otwarcie strumieni *to_host_read_32_1-4* (opisane w rozdziale 3.2.6). Sygnały służą jako dodatkowe zabezpieczenie przed transmisją do użytkownika przechwytywanych ramek. Jeżeli dany strumień nie jest otwarty z poziomu programu komputerowego to mimo tego, że rejestr odpowiedzialny za włączenie przechwytywania jest załączony, to sygnał wyjściowy capture_ctl będzie dalej w stanie niskim.



Rysunek 3.21: Blok kontrolny.

3.2.6 Wymiana danych

Transmisja danych pomiędzy układem FPGA, a komputerem stacjonarnym odbywa się z wykorzystaniem rozwiązania firmy Xillybus (rozdział 2.6.3). Rozwiązanie to wykorzystuje do swojego działania połączenie PCI-Express (rozdział 2.5). Blok został przystosowany pod wymagania projektu poprzez spersonalizowanie i wygenerowanie na stronie internetowej producenta. Z poziomu programu komputerowego, odczyt i zapis danych do/z układu FPGA jest widoczny jako operacja I/O pliku.

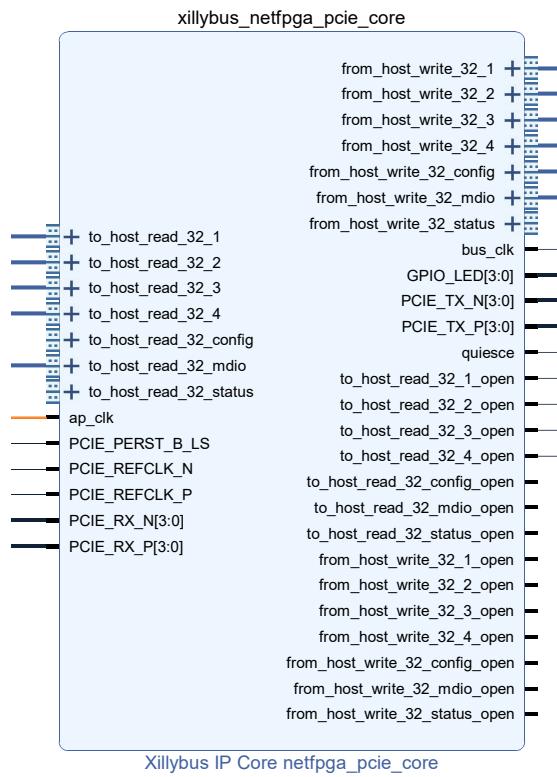
Na rysunku 3.22 przedstawiono rzeczywisty wygląd bloku z programu Vivado. Każdy blok Xillybus posiada sygnały niezależnie od konfiguracji bloku, takie jak:

- Sygnały z przedrostkiem PCIE_ - służące do podłączenia układu pod fizyczne sygnały elektryczne złącza PCI-Express.
- Sygnał wejściowy ap_clk - zegar aplikacji, czyli sygnał zegarowy synchroniczny do wszystkich strumieni wejściowych/wyjściowych typu *AXI4-Stream*.
- Sygnał wyjściowy quiesce - sygnał sygnalizujący, że z poziomu systemu operacyjnego urządzenie jest wyłączone, w projekcie wykorzystywany jako sygnał resetujący układy PHY i wszystkie pozostałe bloki.
- Sygnał wyjściowy bus_clk - zegar referencyjny z gniazda PCI-Express do dowolnego wykorzystania.
- Sygnały wyjściowe GPIO_LED[3:0] - służące do podpięcia wskaźników LED, informujących o pracy połączenia PCI-Express.

Sygnały z przedrostkiem to_host oraz from_host to interfejsy, które zostały dobrane do wymagań projektu. Wszystkie interfejsy do przesyłania danych wykorzystują protokół *AXI4-Stream* o 32-bitowej szerokości szyny:

- to_host_read_32_1-4 - strumienie danych wychodzące do systemu operacyjnego, zawierające przechwycone i przefiltrowane ramki ethernetowe.
- from_host_write_32_1-4 - strumienie danych przychodzące z systemu operacyjnego, zawierające konfiguracje nastaw filtrów modułów.

- to_host_read_32_mdio - strumień danych wychodzący, wykorzystywany w bloku MDIO (rozdział 3.2.4).
- from_host_read_32_mdio - strumień danych przychodzący, wykorzystywany w bloku MDIO (rozdział 3.2.4).
- from_host_read_32_config - strumień danych przychodzący, wykorzystywany w bloku kontrolującym (rozdział 3.2.5).
- to_host_read_32_config, to_host_read_32_status, from_host_read_32_status - niewykorzystane strumienie danych.

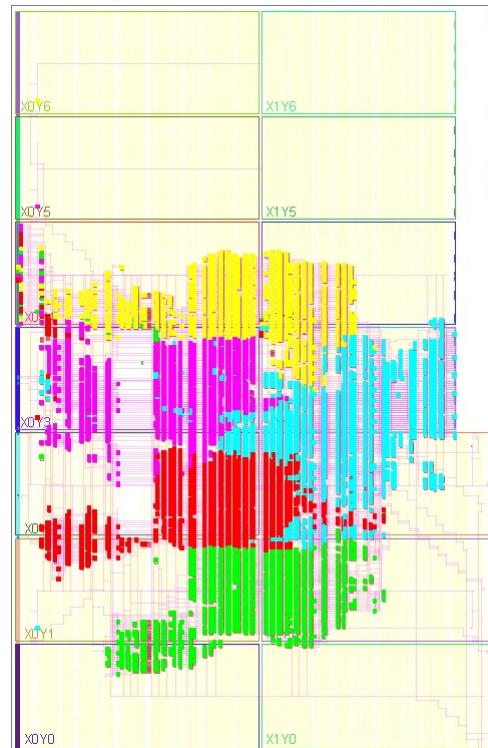


Rysunek 3.22: Spersonalizowany blok Xillybus.

Sygnały z przyrostkiem `_open` sygnalizują otwarcie danego strumienia z poziomu systemu operacyjnego. W projekcie wykorzystywane są tylko sygnały dla otwarcia strumieni `to_host_read_32_1-4`, które są używane w bloku kontrolującym.

3.2.7 Wyniki implementacji projektu sprzętowego

Na rysunku 3.23 przedstawiono fizyczne rozmieszczenie wykorzystanych komórek logicznych na układzie. Kolorem niebieskim zaznaczono blok Xillybus, pozostałymi kolorami oznaczono 4 główne moduły przechwytująco-filtracyjne.



Rysunek 3.23: Zrzut ekranu z programu Vivado przedstawiający graficzne rozmieszczenie poszczególnych modułów.

W tabeli 3.1 znajdują się dane na temat wykorzystania dostępnych zasobów sprzętowych układu FPGA. Można zauważyć, że układ posiada sporo pozostałych komponentów. Oznacza to, że projekt może być dalej rozwijany i rozbudowywany o kolejne funkcje i możliwości.

Tablica 3.1: Tabela przedstawiająca wykorzystanie dostępnych komórek i modułów układu FPGA.

Typ komórki	Wykorzystanych, j.	Dostępnych, j.	Użytych, %
LUT	17882	203800	8.77
LUTRAM	234	64000	0.37
FF	29399	407600	7.21
BRAM	81	445	18.20
IO	61	400	15.25
GT	4	8	50.0
BUFG	28	32	87.5
MMCМ	6	10	60
PLL	1	10	10.0
PCIe	1	1	100.0

3.3 Część programowa

Program komputerowy o nazwie *NetCon* jest nierozerwalną częścią prezentowanego systemu. Program spełnia następujące funkcje:

- odbieranie danych z karty NetFPGA i zapisywanie na dysk (rozdział 3.3.3),
- wysyłanie nastaw do bloków filtracyjnych (rozdział 3.3.4),
- sterowanie otwarciem lub zamknięciem tunelu (rozdział 3.3.5),
- obsługa interfejsu MDIO układów PHY (rozdział 3.3.6),
- możliwość wyłączenia funkcji Energy-Efficient Ethernet (rozdział 3.3.7).

3.3.1 Uzasadnienie wyboru języka programowania oraz interfejsu

Wybór języka C++ oraz interfejsu programu konsolowego został uwarunkowany pewnymi czynnikami.

Język C++ został wybrany głównie z powodu wysokiej wydajności kodu wynikowego oraz stosunkowo łatwej przenośności pomiędzy różnymi systemami i architekturami. Pozwala w optymalny sposób komunikować się z urządzeniami peryferyjnymi. Dobrze zaprojektowany program jest w stanie bardzo dobrze wykorzystać dostępne możliwości sprzętu.

Interfejs konsolowy oraz idea uruchamiania programu z wiersza poleceń została wybrana w związku z łatwością tworzenia zewnętrznych skryptów. Konsolowy interfejs umożliwia wykorzystanie wyłącznie standardowych bibliotek, co w przyszłości może bardzo ułatwić przenoszenie kodu na inny system operacyjny np. Linux.

3.3.2 Komunikacja z blokiem Xillybus

Wymiana danych z blokiem Xillybus z poziomu programu polega na otwarciu pliku o specjalnej nazwie oraz zapisywania lub odczytywania do/z otwartego pliku. Przykładowo, aby odebrać dane ze strumienia `to_host_read_32_1`, należy otworzyć plik o nazwie `\.\xillybus_read_32_1` oraz odczytać określona ilość danych. Pliki urządzenia Xillybus mają swoje ograniczenia w porównaniu do zwykłych plików. Można wyłącznie używać dwóch operacji `_read()` oraz `_write()`.

W programie do tego celu została wykorzystana biblioteka `<io.h>`. Biblioteka na systemie Windows oferuje metody nisko-poziomowe dostępu do plików, takie jak:

- `_open()` - otwiera plik,
- `_read()` - czyta z pliku,
- `_write()` - zapisuje do pliku,
- `_close()` - zamyka plik.

Każdy ze strumieni jest jednokierunkowy, oznacza to, że z pojedynczego strumienia można wyłącznie odczytywać lub zapisywać. Otwarcie strumienia z którego dane będą odczytywane odbywa się poprzez dodanie flagi `_O_RDONLY`, a strumienia do którego dane będą zapisywane flagi `_O_WRONLY`:

```
int devr = _open(deviceName, _O_RDONLY | _O_BINARY)
int devw = _open(deviceName, _O_WRONLY | _O_BINARY)
```

Flagi znajdują się w nagłówku <fcntl.h>. Dodanie flagi _O_BINARY jest potrzebne, aby zapobiec interpretacji wartości bajtów przez system (np. wartości 0x1A jako koniec pliku EOF).

Kolejno po otwarciu pliku dane można odczytywać lub zapisywać za pomocą metod:

```
int rc = _read(devr, buf, ilosc_bajtow)
int wc = _write(devw, buf, ilosc_bajtow)
```

3.3.3 Przechwytywanie - *capture*

Funkcja *capture* rozpoczyna odbieranie i zapisywanie przechwytywanego ruchu sieciowego z danego portu do pliku. Wywołanie funkcji odbywa się z wiersza poleceń z następującymi argumentami:

```
NetCon.exe capture N F S
```

Gdzie,

- N - numer portu/modułu,
- F - nazwa pliku lub ścieżka (opcjonalna),
- S - rozmiar bufora cyklicznego w megabajtach (opcjonalna).

Przykład minimalnego wywołania:

```
NetCon.exe capture 1
```

Implementacja

Program po inicjalizacji bufora cyklicznego, utworzenia pliku z nagłówkiem *PCAP* i otwarcia strumienia danych z karty, tworzy cztery wątki:

- wątek odczytujący dane ze strumienia (pliku) \\.\xillybus_read_32_N,
- wątek zapisujący do pliku *PCAP*,

- wątek monitorujący,
- wątek czekający na wcisnięcie klawisza.

Po utworzeniu powyższych wątków następuje wysłanie rejestru otwierającego przechwytywanie do bloku kontrolującego (opisanego w rozdziale 3.2.5). Taki mechanizm zapewnia synchronizację i kompletność odbieranych danych.

Wątek odczytujący bezpośrednio odczytuje dane z otwartego strumienia \\.\xillybus_read_32_N do bufora cyklicznego. Wątek zapisujący bezpośrednio zapisuje dane z bufora cyklicznego do pliku. Wątek monitorujący zlicza liczbę bajtów w buforze cyklicznym i wyświetla podstawowe informacje na temat ilości zapisanych danych, przepustowości, przepełnieniach bufora cyklicznego oraz maksymalnym zużyciu pamięci bufora cyklicznego, co zostało pokazane na rysunku 3.24.

```

PS C:\netfpga_backup\testy> .\NetCon.exe c 1
Alokowanie bufora o rozmiarze 16777216 bajtów
Otwarcie strumienia NetFPGA \\.\xillybus_read_32_1 powiodło się.

Naciśnij klawisz x aby zatrzymać przechwytywanie i zamknąć program.
Licznik           Przepustowość          Pełny bufor          Maks. zajetosc bufora
0                 0.000000 MB/s          0 razy             0 bajtów
  
```

Rysunek 3.24: Program w trybie monitoringu ruchu sieciowego z zapisem do pliku.

Ostatni wątek czeka na naciśnięcie klawisza “x”. W momencie naciśnięcia klawisza, następuje zmiana wartości zmiennej globalnej, która informuje wątki o rozpoczęciu procesu zamknięcia programu. Zamknięcie programu rozpoczyna się od wysłania do bloku kontrolującego rejestru, który zatrzymuje przechwytywanie ramek sieciowych na karcie NetFPGA. Kolejno program czeka 100ms na odebranie pozostałych danych, które znajdują się w buforach nadawczych/odbiorczych oraz kończy działanie programu.

3.3.4 Nastawy filtrów - *set*

Funkcja *set* służy do wysłania nastaw filtrów dla danego bloku filtracyjnego. Wywołanie funkcji odbywa się z wiersza poleceń z następującymi argumentami:

```
NetCon.exe set N M1 F1 M2 F2 M3 F3 M4 F4
```

Gdzie,

- N - numer portu/bloku filtrującego,
- M1 - M4 - minimalna długość ramki (wliczając preambułę) dla 1 - 4 filtra,
- F1 - F4 - nastawy filtra 1 - 4.

Przykład wysłania nastaw filtrów na porcie 1 z przepuszczaniem wszystkich ramek do użytkownika:

```
NetCon.exe set 1 0
```

Format nastaw dla pojedynczego filtra wygląda następująco:

```
"NB_1 A_1 B_1;NB_2 C_2;NB_3 A_3 B_3;..."
```

Gdzie,

- NB - numer bajtu ramki (wliczając preambułę),
- A - liczba od której porównywany bajt ma być większy lub równy,
- B - liczba od której porównywany bajt ma być mniejszy lub równy,
- C - liczba, której porównywany bajt ma być równy.

Przykład wywołania:

```
NetCon.exe set 1 80 "9 192 193;10 25;11 120 150;"
```

Powyższe argumenty i nastawy filtra oznaczają, że:

- nastawy wysłane mają być do bloku filtracyjnego 1 portu,
- minimalna długość ramki ma wynosić 80 bajtów,
- wartość bajtu ramki na pozycji 9 ma być w przedziale <192,193>,
- wartość bajtu na pozycji 10 ma być równa wartości 25,
- wartość bajtu na pozycji 11 ma znajdować się w przedziale <120;150>.

Wartości można zapisywać w formacie heksadecymalnym (z prefiksem 0x np. 0xF9), decymalnym i oktagonalnym (z prefiksem 0, np. 012). W przypadku nastaw, należy pamiętać, że system przechwytuje całą ramkę włącznie z preambułą. Rzeczywista ramka zaczyna się w takim przypadku po 8 bajtach. Oznacza to, że pierwszy bajt ramki ethernetowej zaczyna się na indeksie 8.

3.3.5 Sterowanie tunelami - *bridge*

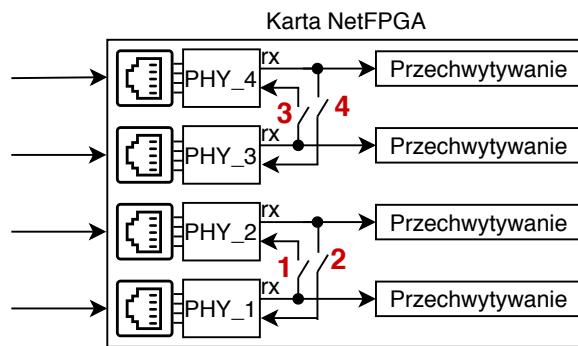
Funkcja *bridge* oferuje możliwość wyłączania oraz załączania wybranego tunelu. Wywołanie funkcji odbywa się z wiersza poleceń z następującymi argumentami:

```
NetCon.exe bridge_on N
NetCon.exe bridge_off N
NetCon.exe bridge_all
```

Gdzie,

- N - numer tunelu.

Funkcja *bridge_on* załącza określony tunel, funkcja *bridge_off* wyłącza określony tunel, a funkcja *bridge_all* załącza wszystkie tunele. Na rysunku 3.3.5 została przedstawiona numeracja tuneli.



Rysunek 3.25: Schemat przedstawiający numerację tuneli.

Przykłady wywołania:

```
NetCon.exe bridge_on 1
NetCon.exe bridge_off 2
```

```
NetCon.exe bridge_all
```

3.3.6 Rejestry MDIO - *mdio*

Funkcja *mdio* umożliwia zapis oraz odczyt rejestrów układów PHY. Dokładny opis poszczególnych wartości interfejsu znajduje się w rozdziale 2.1.3. Wywołanie funkcji odbywa się z wiersza poleceń z następującymi argumentami:

```
NetCon.exe mdio op_code phy_addr reg_addr data
```

Po wywołaniu programu z określonymi argumentami, wyświetlana jest wysłana oraz zwrócona wartość pola danych. Przykładowy odczyt (*op_code* = 2) z układu PHY o numerze 1 (*phy_addr* = 1), rejestrów 0 i 1 (*reg_addr* = 0 i 1) przedstawiono na rysunku 3.26.

```
PS C:\netfpga_backup\testy> .\NetCon.exe mdio 2 1 0
Wysyłanie po MDIO: 0x60820000 Data: 0b00000000000000000000
Odebrano po MDIO: 0x60821140 Data: 0b0001000101000000
PS C:\netfpga_backup\testy> .\NetCon.exe mdio 2 1 1
Wysyłanie po MDIO: 0x60860000 Data: 0b00000000000000000000
Odebrano po MDIO: 0x60867969 Data: 0b0111100101101001
```

Rysunek 3.26: Zrzut ekranu programu po wywołaniu funkcji *mdio*.

3.3.7 Opcja EEE - *eee_off*

W programie została zaimplementowana funkcja *eee_off*, która wyłącza opcję *Energy-Efficient Ethernet* na wszystkich układach PHY poprzez interfejs MDIO. Wyłączenie opcji polega na wysłaniu odpowiednich rejestrów dla każdego z układu PHY. Wywołanie funkcji odbywa się z wiersza poleceń z następującym argumentem:

```
NetCon.exe eee_off
```

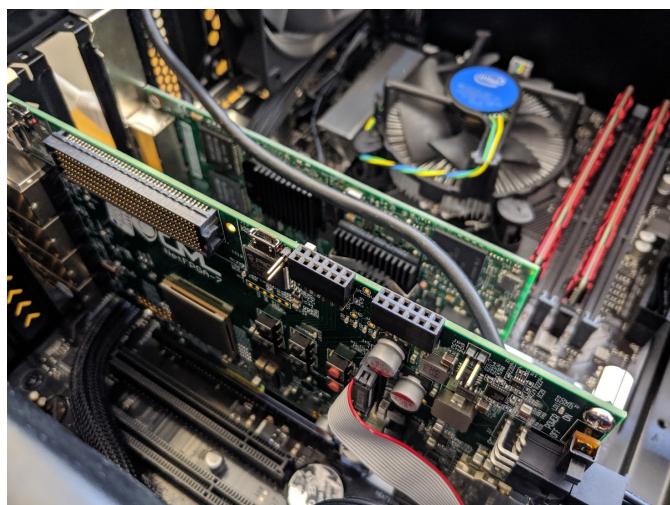
3.4 Uruchomienie i testowanie

Uruchomienie prezentowanego systemu należy zacząć od włożenia karty NetFPGA-1G-CML w gniazdo PCI-Express o szerokości 4x lub większej do odłączonego od zasilania komputera. Kolejno podłączyć 6-pinową wtyczkę zasilającą PCIE do karty

NetFPGA-1G-CML oraz podłączyć programator firmy Xilinx. Wygląd stacji roboczej z włożoną kartą przedstawiono na rysunkach 3.27 i 3.28.

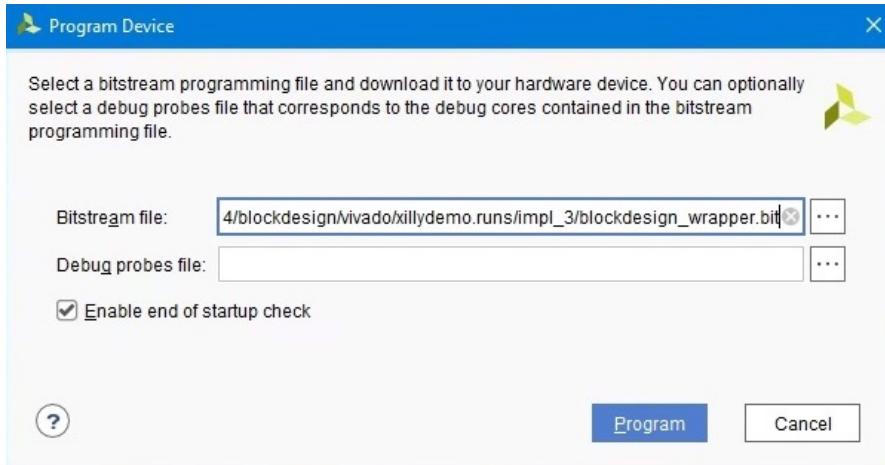


Rysunek 3.27: Wygląd stacji roboczej z włożoną kartą NetFPGA-1G-CML.



Rysunek 3.28: Wygląd włożonej karty NetFPGA-1G-CML z podłączonym programatorem z bliska.

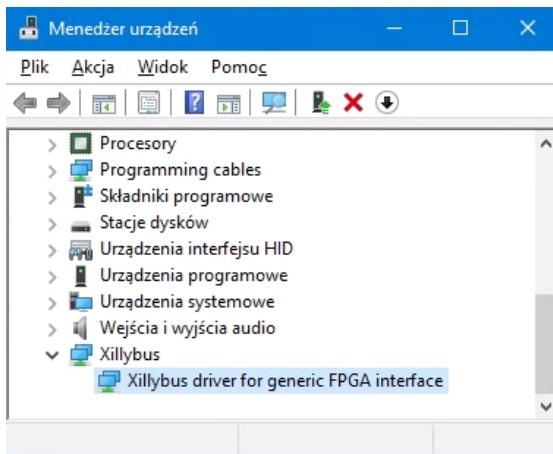
Następnie należy załączyć komputer oraz uruchomić środowisko Vivado. Kolejno otworzyć projekt oraz wgrać plik wynikowy implementacji tzw. bitstream. Rysunek 3.29 przedstawia okno wyboru pliku w programie Vivado.



Rysunek 3.29: Zrzut ekranu z programu Vivado przed wgraniem tzw. bitstream'a.

Po operacji wgrania bitstream'a należy ponownie uruchomić komputer, ponieważ płyta główna musi wykryć nowe urządzenie PCI-Express. Alternatywną opcją jest wgranie projektu prosto na pamięć nieulotną BPI znajdująca się na karcie. Taka opcja zapewnia załadowanie projektu na układ FPGA przy każdym załączaniu zasilania, dzięki czemu po wyłączeniu zasilania komputera projekt nie zostanie utracony.

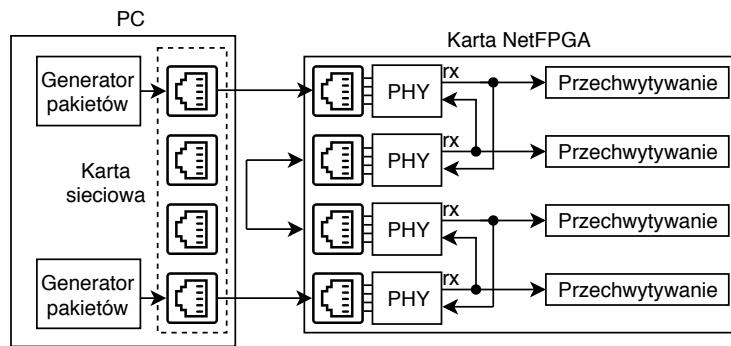
Następnie, jeżeli sterownik firmy Xillybus nie został zainstalowany należy wskazać urządzenie oraz wgrać sterownik ręcznie. Jeżeli wszystkie etapy zostały wykonane poprawnie, w oknie menedżera urządzeń powinno znajdować się urządzenie *Xillybus driver for generic FPGA interface*, tak jak przedstawiono na rysunku 3.30. Od tego momentu system jest gotowy do działania.



Rysunek 3.30: Zrzut ekranu z okna menedżera urządzeń.

3.4.1 Testowanie

Kolejnym krokiem w przetestowaniu urządzenia jest podłączenie przewodów ethernetowych do odpowiednich portów karty NetFPGA i karty sieciowej. W tym rozdziale zostanie opisana konfiguracja wykorzystująca wszystkie porty karty NetFPGA. Przedstawiona konfiguracja została wykorzystana w rozdziale 4.2 do pomiaru opóźnień wprowadzanych przez układy PHY. Na rysunku 3.31 znajduje się schemat połączeniowy, a na rysunku 3.32 rzeczywiste zdjęcie połączeń.



Rysunek 3.31: Konfiguracja pozwalająca na pomiar opóźnień tunelu.



Rysunek 3.32: Zdjęcie przedstawiające podłączenie w konfiguracji umożliwiającej pomiar opóźnień tunelu. (Na dole karta NetFPGA, nad nią karta sieciowa).

Następnie należy otworzyć cztery okna konsolowe oraz uruchomić program z następującymi argumentami:

```
NetCon.exe capture 1  
NetCon.exe capture 2  
NetCon.exe capture 3  
NetCon.exe capture 4
```

Otwarcie strumienia z powyższą komendą powoduje utworzenie się pliku o rozszerzeniu .pcap z datą i godziną uruchomienia programu oraz numerem portu. Struktura nazwy plików została pokazana na rysunku 3.33.

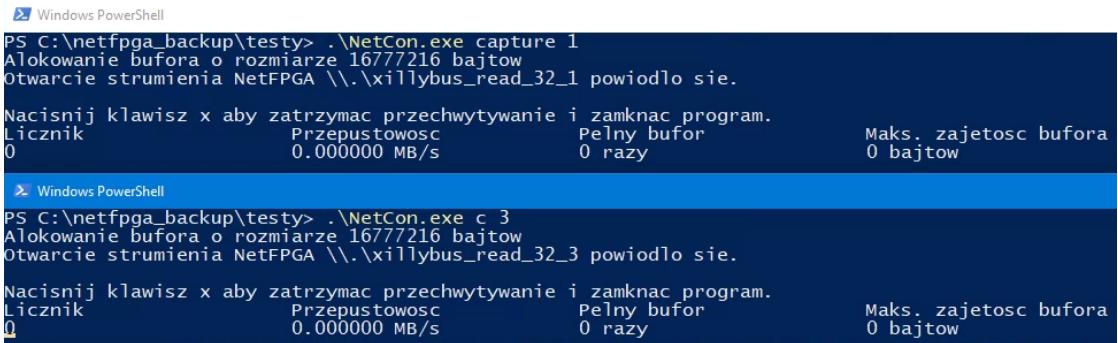
capture_2019_10_1_11_54_9_port1.pcap	01.11.2019 11:54	Wireshark capture...
capture_2019_10_1_11_54_11_port3.pcap	01.11.2019 11:54	Wireshark capture...

Rysunek 3.33: Struktura nazewnictwa zapisywanych plików.

Istnieje także możliwość zdefiniowania własnej nazwy pliku poprzez dodanie kolejnego argumentu do wywołania programu:

```
NetCon.exe capture 1 "nazwa_pliku.pcap"
```

Zrzut ekranu z programów uruchomionych w trybie przechwytywania został przedstawiony na rysunku 3.34.



```

Windows PowerShell
PS C:\netfpga_backup\testy> .\NetCon.exe capture 1
Alokowanie bufora o rozmiarze 16777216 bajtow
Otwarcie strumienia NetFPGA \\.\xillybus_read_32_1 powiodlo sie.

Nacisnij klawisz x aby zatrzymac przechwytywanie i zamknac program.
Licznik           Przepustosc          Pusty bufor      Maks. zajetosc bufora
0                 0.000000 MB/s        0 razy            0 bajtow

Windows PowerShell
PS C:\netfpga_backup\testy> .\NetCon.exe c 3
Alokowanie bufora o rozmiarze 16777216 bajtow
Otwarcie strumienia NetFPGA \\.\xillybus_read_32_3 powiodlo sie.

Nacisnij klawisz x aby zatrzymac przechwytywanie i zamknac program.
Licznik           Przepustosc          Pusty bufor      Maks. zajetosc bufora
0                 0.000000 MB/s        0 razy            0 bajtow

```

Rysunek 3.34: Zrzut ekranu z programów w trybie przechwytywania i zapisywania ruchu sieciowego do pliku.

Nastawy filtrów w bloku filtracyjnym na karcie NetFPGA początkowo są ustalone w trybie blokowania wszystkich ramek. Aby ustawić wszystkie filtry na wszystkich portach w trybie przekazywania całego ruchu sieciowego do użytkownika, należy uruchomić program z następującymi argumentami:

```

NetCon.exe set 1 0
NetCon.exe set 2 0
NetCon.exe set 3 0
NetCon.exe set 4 0

```

Dodatkowo wyłączone są wszystkie tunele, oznacza to, że domyślnie odbierane ramki na układach PHY nie są przekazywane do przeciwnego układu PHY. Aby załączyć wszystkie ścieżki, należy uruchomić program z następującym argumentem:

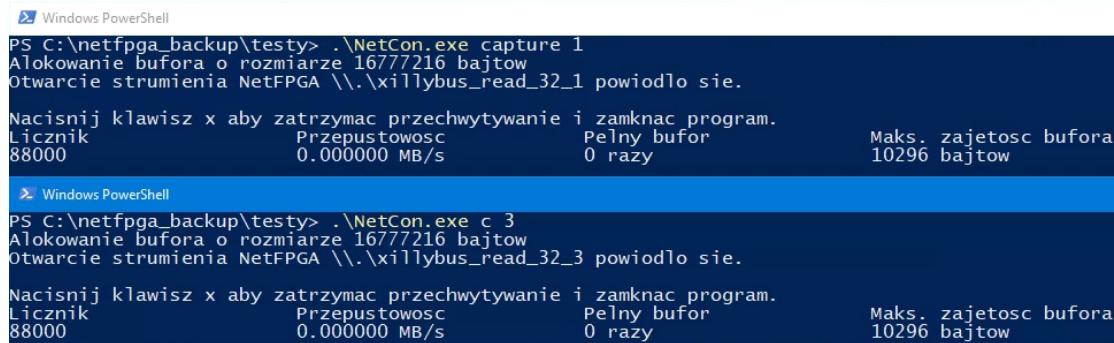
```
NetCon.exe bridge_all
```

Ostatnim krokiem jest wygenerowanie ramek na jednym z portów karty sieciowej. Można to zrobić za pomocą pythonowego skryptu z wykorzystaniem biblioteki Scapy:

```
from scapy.all import *
```

```
sendp("abcdefghijklmнопqrstuvwxyz", iface=ifaces.  
    ↪ dev_from_pcapname('\\\\Device\\\\NPF_{DB07413B-64B8  
    ↪ -4005-A7B3-9F13EAB5210E}') , count=1000)
```

Na rysunku 3.35 zauważać można, że wszystkie ramki zostały przechwycone. Licznik określa liczbę bajtów odebranych z danego strumienia.

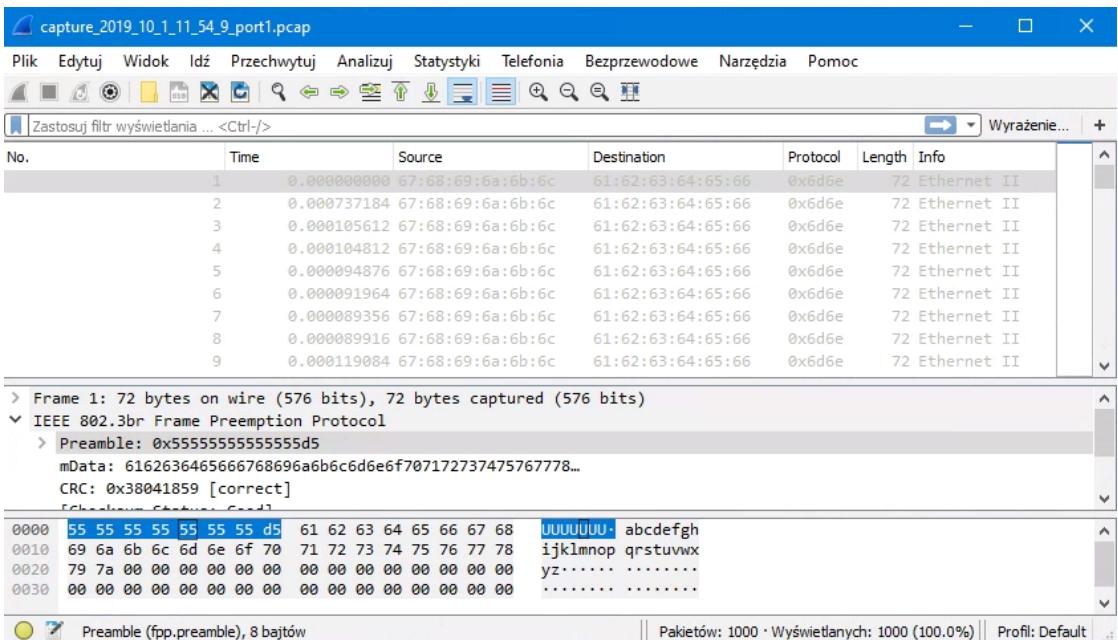


The screenshot shows two separate Windows PowerShell windows. Both windows display the command `.\NetCon.exe capture 1` and the output of the NetFPGA driver's configuration. The first window shows the configuration for stream `xillybus_read_32_1`. The second window shows the configuration for stream `xillybus_read_32_3`. Both windows include a table with the following data:

Licznik	Przepustowosc	Pelny bufor	Maks. zajetosc bufora
88000	0.000000 MB/s	0 razy	10296 bajtow

Rysunek 3.35: Zrzut ekranu z programu po wygenerowaniu ramek.

Kolejno plik można otworzyć programem Wireshark oraz zobaczyć, że wszystkie dane są poprawne oraz liczba przechwyconych ramek zgadza się z liczbą wysłanych. Zrzut ekranu z programu został przedstawiony na rysunku 3.36.

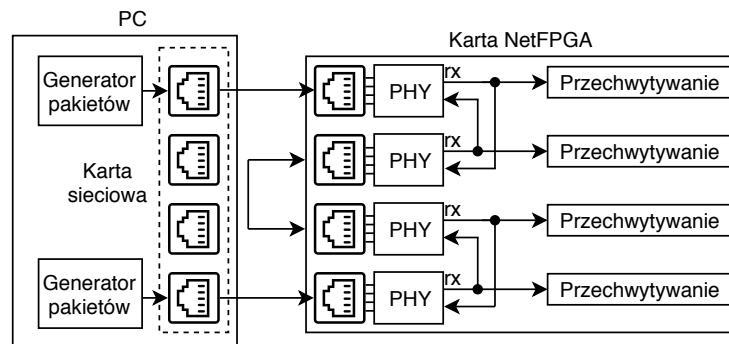


Rysunek 3.36: Zrzut ekranu po otwarciu pliku przechwytywania w programie Wireshark.

3.4.2 Wpływ trybu Energy-Efficient Ethernet na opóźnienia

W trakcie pierwszych pomiarów w konfiguracji przedstawionej na rysunku 3.37 okazało się, że układy PHY, które znajdują się na karcie NetFPGA mają domyślnie załączoną opcję Energy-Efficient Ethernet. Powoduje to usypanie się układów PHY, które wprowadza dodatkowe opóźnienia w tor transmisji. Maksymalne zmierzzone opóźnienie wynosiło 18.9 μ s, a minimalne 0.8 μ s. Specyfikacja Energetyczno-Efektywnego Ethernetu [10] mówi o wartości 16.5 μ s w przypadku 1000BASE-T, jako najgorszym możliwym czasie wybudzenia układu PHY. Zmierzzone opóźnienie jest większe o 2.4 μ s. Może to być spowodowane wewnętrznymi buforami układów. Oczywiście istnieje możliwość wyłączenia opcji ustawiając odpowiedni rejestr w układach PHY przez interfejs MDIO. Po wyłączeniu ustawienia oraz miękkim restartie modułów PHY wartość opóźnień spadła do 0.7 μ s.

Działanie portu w tym trybie można wykryć poprzez inspekcję wskaźników



Rysunek 3.37: Konfiguracja pozwalająca na pomiar opóźnień.

LED przy portach karty NetFPGA. Jeżeli układ wynegocjował wspomniany tryb, a połączenie zostało ustalone, to wskaźnik LED przy danym porcie będzie migać naprzemienne, będąc 2 s w stanie wyłączonym oraz 0.4 s w stanie załączonym. Przy pomiarach należy zwrócić na to uwagę, ponieważ mogą zaburzyć dokładność wyników.

Rozdział 4

Badania

4.1 Analiza zdolności przechwytywania

Wiele sposobów na monitoring ruchu sieciowego wymaga komputera z kartą sieciową. Przełącznik sieciowy z klonowaniem portów, niektóre modele TAP'ów wymuszają przechwytywanie ruchu z wykorzystaniem interfejsu ethernetowego. Najsłabszym miejscem w takich zestawieniach jest zazwyczaj komputer, który wymaga programowego przetwarzania pakietów od strony systemu operacyjnego i programu. W tym rozdziale zostanie porównany narzut i stabilność takiego przechwytywania w porównaniu do opisywanego w pracy systemu.

4.1.1 Stanowisko badawcze

Stanowisko składało się z dwóch elementów: urządzenia przechwytyjącego jakim był komputer z kartą sieciową i kartą NetFPGA oraz laptopa, który służył jako generator ramek ethernetowych.

Testowanie opierało się o porównanie ogólnodostępnych rozwiązań programowych: *Tcpdump* oraz *Wireshark* (GUI i Dumpcap) z rozwiązaniem sprzętowym w postaci prezentowanego w pracy systemu.

Główny komputer pracował w konfiguracji:

- System operacyjny - Windows 10 Education, wersja 1903;

- Procesor - Intel Core i5-8400, 6 rdzeni z wymuszonym stałym taktowaniem 2.8Ghz;
- Pamięć RAM - DDR4-2666 2x4GB pracująca w trybie Dual-Channel;
- Dysk SSD - PLEXTOR PX-512M9PeG;
- Karta sieciowa - Intel Gigabit ET Quad Port;

Generatorem ramek sieciowych był laptop o konfiguracji:

- System operacyjny - Ubuntu 16.04;
- Procesor - Intel Core i7-8550u;
- Pamięć RAM - DDR4 2x4GB pracująca w trybie Dual-Channel;
- Karta sieciowa - TP-Link UE330;

4.1.2 Metodyka pomiarowa

Podczas testów mierzono 4 parametry. Dla każdej z 5 prób sprawdzano liczbę odebranych pakietów. Całkowicie dla każdego wariantu testowego mierzono 3 parametry z pół-sekundowym interwałem, takie jak: obciążenie pojedynczego wątku, obciążenie procesora oraz pobór energii procesora. Liczbę odebranych ramek odczytywano z programów przechwytyujących, a pozostałe 3 parametry z programu HWiNFO [9]. Liczbę odebranych ramek uśredniono oraz wyliczono procent utraconych. Dane z programu HWiNFO zostały przefiltrowane średnią ruchomą z 5 ostatnich okresów, aby zniwelować szum pomiarowy. Kolejno wyszukiwane były maksymalne wartości przefiltrowanych danych jako wskaźniki pomiarowe danej metody.

Pomiar składał się z 5 krotkiego wykonania testu. Pojedynczy test składał się z wygenerowania przez program tragen jednego z wariantu:

- 15625000 ramek o długości 68 bajtów,
- 1953125 ramek o długości 516 bajtów,
- 660502 ramek o długości 1518 bajtów.

Ze względu na programową naturę generatora ruchu sieciowego, łączne nie zostało w 100% nasycone, jednakże stopień obciążenia łącza dochodził w każdym przypadku do ponad 90%.

4.1.3 Wyniki

W tabeli 4.1 zostały przedstawione wyniki pomiarów. Jak można zauważyć, każde z rozwiązań programowych nie zdołało przechwycić całego wygenerowanego ruchu sieciowego. Najlepszym z rozwiązań programowych okazał się program *Dumpcap*, który jest dostarczany razem z pakietem *Wireshark*.

Tablica 4.1: Wyniki pomiarów zdolności przechwytywania przy wysokim obciążeniu łącza.

Metoda	Dług. ramki	Maks. obciążenie jednego wątku, %	Maks. obciążenie procesora, %	Maks. pobór energii procesora, Wat	Utraconych pakietów, %
Dumpcap	68	99	31	37	5.061
	516	52	15	26	0.506
	1514	37	9	24	0.001
Tcpdump	68	96	30	36	80.615
	516	96	23	32	32.339
	1514	62	15	27	0.272
WiresharkD	68	97	48	45	74.160
	516	99	35	38	10.428
	1514	74	21	30	0.001
NetFPGA	68	7	3	21	0.000
	516	8	4	20	0.000
	1514	7	3	20	0.000
Bezczynność		4	2	20	

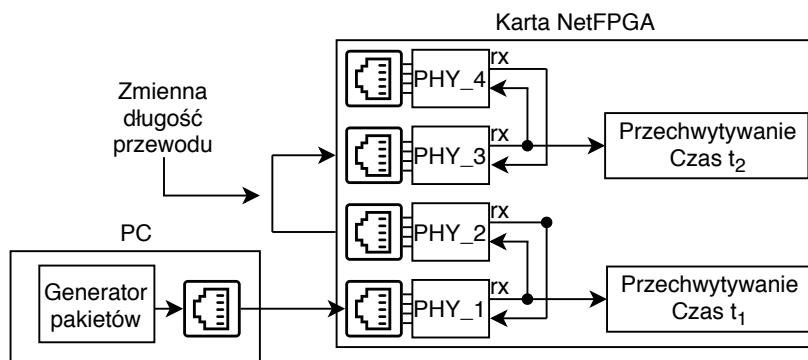
System sprzętowy w porównaniu do rozwiązań programowych pozwala na znaczną redukcję obciążenia procesora oraz poboru energii. Dodatkowo, narzut na system operacyjny jest niewielki i praktycznie niezależny od natężenia i długości odbieranych ramek. Przewaga rozwiązania sprzętowego polega na ominięciu części sieciowej systemu operacyjnego, przez którą w przypadku rozwiązań programowych musi przechodzić cały ruch sieciowy.

4.2 Eksperyment precyzji pomiarowej

Zaprezentowany w pracy system może służyć do pomiaru opóźnień wprowadzanych przez testowane urządzenie sieciowe. Eksperyment ma na celu pokazanie możliwości układu oraz precyzji znaczników czasowych dodawanych do każdej ramki na układzie FPGA, a także określenie opóźnień wprowadzanych przez sam układ pomiarowy.

4.2.1 Stanowisko badawcze

W badaniu wykorzystano konfigurację przedstawioną na rysunku 4.1, gdzie parametrem była długość przewodu symulująca opóźnienia wprowadzane przez testowane urządzenie.



Rysunek 4.1: Konfiguracja pozwalająca na pomiar opóźnień.

4.2.2 Metodyka pomiarowa

Procedura testowa składała się z jednorazowego wygenerowania 1000 ramek. Mierząc czas przyjścia ramki w dwóch miejscach na porcie 1 oraz 3, możliwe jest obliczenie opóźnienia wprowadzanego przez system. Czas opóźnienia składa się z:

$$t_{diff} = t_2 - t_1 = t_{phyout} + t_o + t_{phyin} \quad (4.1)$$

Gdzie,

- t_{diff} - sumaryczne opóźnienie wprowadzane przez układy PHY oraz przewód,
- t_2 - znacznik czasowy nadania ramki na porcie 3,
- t_1 - znacznik czasowy nadania ramki na porcie 1,
- t_{phyout} - czas konwersji z interfejsu RGMII na sygnały elektryczne układu PHY,
- t_o - czas propagacji dla przewodu zależny od długości,
- t_{phyin} - czas konwersji z sygnałów elektrycznych na interfejs RGMII układu PHY.

Zakładając, że bardzo krótki przewód wprowadza opóźnienie t_o dużo mniejsze od rozdzielczości pomiarowej wynoszącej w tym przypadku 4 ns oraz, że opóźnienia wprowadzane przez układy PHY są stałe, to można w takim przypadku pominąć wartość t_o i obliczyć sumaryczny czas wprowadzanych opóźnień przez układy PHY. Odejmując wyznaczoną wartość opóźnienia od zmierzonej różnicy t_{diff} jest możliwe wyznaczenie wartości t_o w przypadku pomiaru dłuższych przewodów (lub urządzenia sieciowego):

$$t_o = t_{diff} - (t_{phyout} + t_{phyin}) \quad (4.2)$$

Czas propagacji sygnałów elektrycznych w przewodzie o danej długości można obliczyć ze wzoru [5]:

$$t_o = \frac{l}{c \cdot NVP}. \quad (4.3)$$

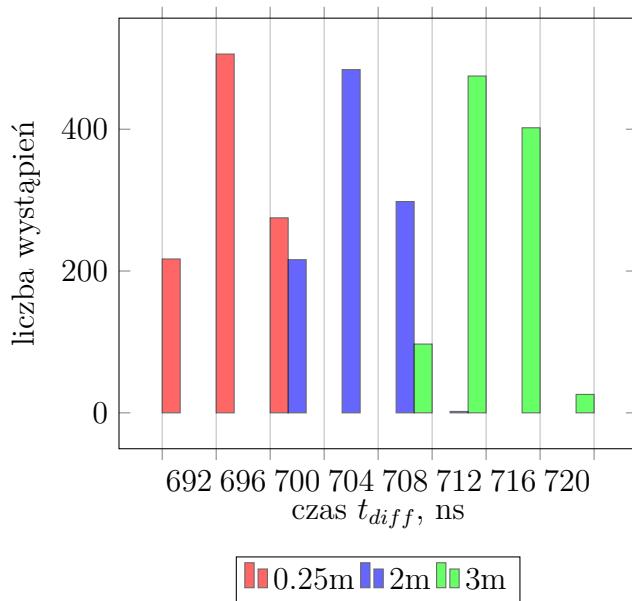
Gdzie,

- l - długość przewodu,
- c - prędkość światła w próżni,
- NVP - współczynnik propagacji dla danego medium - NVP (ang. *Nominal Velocity of Propagation*).

Przykładowo dla przewodu o długości 1 m oraz współczynniku NVP wynoszącym 69%, czas propagacji wynosi 4.83 ns.

4.2.3 Wyniki

Na rysunku 4.2 został przedstawiony histogram z wartościami opóźnień. Jak można zauważyć opóźnienie rośnie i jest zależne od długości przewodu. Rozstęp pomiarów dla badanej konfiguracji wynosił do 12ns.



Rysunek 4.2: Histogram opóźnień w zależności od długości przewodu.

W tabeli 4.2 w przedostatniej kolumnie zostały estymowane długości przewodu. Wartości zostały obliczone na podstawie zmierzonych czasów t_{diff} , przyjmując stałą 696.2 ns jako łączne opóźnienie układów PHY oraz współczynnik NVP równy

69%. Łączne opóźnienie układów PHY zostało przyjęte jako średnia z wartości $t_{diff} - t_o$ dla mierzonych przewodów.

Tablica 4.2: Wyniki eksperymentu pomiaru czasu propagacji.

Długość przewodu, l , m	Czas propagacji t_o , ns	Zmierzona wartość średnia t_{diff} , ns	$t_{diff} - t_o$, ns	Estymowana długość przewodu l_e , m	Błąd $l_e - l$, m
0.25	1.2	696.23	695.02	0.00	-0.25
2	9.7	704.34	694.68	1.68	-0.32
3	14.5	713.43	698.93	3.56	0.56

Eksperyment ten potwierdza dużą precyzyję układu pomiarowego, która pozwala nie tylko na zauważenie zmiany długości przewodu, a także na jego estymację. Rozdzielcość znaczników czasowych wynosi 4 ns, a więc minimalna rozpoznawalna zmiana długości wynosi 0.83 m dla NVP 69%. Oznacza to, że dokładność systemu jest ograniczona przez tą wartość. Wyniki eksperymentu udowadniają, że układy PHY wprowadzają stałe opóźnienie, które jest możliwe do estymacji z dokładnością do dziesiątek nanosekund.

4.3 Analiza czasu filtracji

W tej części porównywany będzie czas potrzebny na przefiltrowanie ramki w dwóch wariantach CPU oraz FPGA. Do pomiaru wydajności filtracji na CPU użyto jednowątkowego programu *tcpdump*. Pierwotnie do tego celu miał być wykorzystany program *tshark*, czyli konsolowa wersja programu *Wireshark*, jednak okazało się, że program jest kilkukrotnie wolniejszy od programu *tcpdump*.

4.3.1 FPGA

Sposób działania modułu filtracyjnego na układzie FPGA sprawia, że przepustowość w tym przypadku jest deterministyczna i możliwa do obliczenia. Blok

filtracyjny w testowanym systemie działa z częstotliwością 62.5 MHz i przetwarza ramki strumieniowo. Tak jak w przypadku filtracji z użyciem CPU, moduł sprzętowy przetwarza już spakietyzowany do formatu *PCAP* strumień danych. Oznacza to, że oprócz samej ramki ethernetowej, przetwarzany jest także dodany nagłówek. Jest to wymagane, ponieważ w inny sposób nie byłoby możliwości rozróżnienia ramek w ciągłym strumieniu danych (brak informacji o początku ramki lub długości).

Blok filtracyjny (opisany w rozdziale 3.2.3) podczas jednego cyklu jest w stanie porównać 4 bajty ramki ethernetowej do nastaw 4 filtrów. Każdy filtr ma 28 pól konfiguracyjnych w których można określić przedział wartości bajtu oraz pozycję. Pola konfiguracyjne w pojedynczym filtrze mają ograniczenie w postaci braku możliwości porównywania tych samych pozycji. Oznacza to, że każde pole sprawdza jedną unikalną dla filtra pozycję bajtu w ramce.

W czasie jednego cyklu wykonywanych jest 32 komparacji (4 filtry * 4 bajty ramki * 2 porównania (większe_lub_równe/mniejsze_lub_równe)). Oznacza to, że blok filtracyjny teoretycznie jest w stanie wykonać $32 \cdot 62.5 \text{ MHz} = 2 \frac{\text{mld}}{\text{s}}$ porównań pomijając przetwarzanie nagłówka.

Ze względu na strukturę potokową bloku filtracyjnego, czas przetwarzania ramki jest zależny od jej długości. Pomijając nagłówek ramki i przerwy pomiędzy ramkami, 64 bajtową ramkę blok jest w stanie przefiltrować w 16 cykli. Jeden cykl trwa $\frac{1}{62.5 \text{ MHz}} = 16 \text{ ns}$, oznacza to, że przetwarzanie 64-bajtowej ramki zajmuje $16 \text{ ns} \cdot 16 = 256 \text{ ns}$. Jest to wartość niezależna od liczby nałożony nastaw filtrów/filtrów. Pomijając stały czas przetwarzania nagłówka ramki, czas przetwarzania można wyrazić za pomocą funkcji $t = \text{zaokr.do.góry}(\frac{L}{4}) \cdot 16 \text{ ns}$, gdzie L to długość ramki w bajtach.

4.3.2 CPU

Do pomiarów czasów filtracji z użyciem programu *tcpdump* wykorzystano komputer o następujących parametrach:

- System operacyjny - Ubuntu 18.04.3 LTS, Kernel: Linux 4.15.0-66-generic;
- Procesor - Intel Core i7-4790K ze stałym taktowaniem 4.4GHz;

- Płyta główna - MSI Z87-G45;
- Pamięć RAM - DDR3-1866 4x4GB pracująca w trybie Dual-Channel;
- Dysk SSD - GOODRAM Iridium Pro 480GB;

Przed pomiarami pobrane oraz skompilowane zostały najnowsze wersje kodu źródłowego libpcap-1.9.1 oraz tcpdump 4.9.3. Przed instalacją tcpdump, należało najpierw zainstalować libpcap. Kompilację oraz instalację obydwu programów wykonano według dołączonej ze źródłami instrukcji za pomocą poniższych poleceń:

Kompilacja libpcap:

```
sudo apt install flex
sudo apt install bison
./configure
make
sudo make install
```

Kompilacja tcpdump:

```
./configure
make
sudo make install
```

Zmierzona została także przepustowość odczytu 1 GB pliku. Do tego celu wykorzystano komendę *dd*. Z załączonym cachowaniem pomiary wyniosły 7.9 GB/s przy rozmiarze bufora 16 kB oraz 10.9 GB/s przy rozmiarze bufora 256 kB. Pomiar wykonano w terminalu następującym poleceniem:

```
dd if=1GB_0064_15625000.pcap of=/dev/zero bs=16k
```

Metodyka wykonywanych pomiarów

Czas wykonania filtracji w programie *tcpdump* mierzono z użyciem komendy *time*, która jest standardowo dostępna na systemach typu Linux. Precyzja pomiaru wynosi 0.001 s. Czas wykonania programu mierzono 100-krotnie dla 7 plików w konfiguracji z 1, 2, 4, 8 i 16 nastawami. Wielokrotny pomiar wykonywano za pomocą następującego skryptu:

```
#!/bin/bash

filtr="ether [0]==0||ether [1]==0"
out_log="out6.log"
echo $filtr
echo $out_log

echo "Filtr:$filtr" > $out_log
TIMEFORMAT=$'%R\t%U\t%S'

for file in ./pcap/*
do
    echo "Plik:$file" >> $out_log
    for i in {1..100}
    do
        time(tcpdump -r $file -w /dev/null $filtr &> /
            ↪ dev/null)2>>$out_log
        echo $i
    done
done
```

Zbiór danych

Zestaw danych testowych składa się z 7 plików *PCAP* o wielkości 1 GB. Każdy z plików posiada powielony jeden rodzaj ramki o długości kolejno: 64, 128, 256, 512, 1024, 1280, 1518 bajtów.

Metodyka analizy wyników

Czas filtracji pojedynczej ramki należało by estymować w następujący sposób:

$$\bar{t}_f = (\bar{t}_p - t_o - t_z)/k \quad (4.4)$$

Gdzie,

- \bar{t}_f - uśredniony czas filtracji pojedynczej ramki,
- \bar{t}_p - uśredniony czas wykonania programu,
- t_o - narzut odczytu pliku,
- t_z - narzut zapisu pliku,
- k - liczba ramek w pliku.

Nastawy filtra programu zostały tak dobrane, aby odrzucać wszystkie pakiety. Takie rozwiązanie powoduje, że w obliczeniach można pominąć kwestię czasu potrzebnego na zapis pliku wynikowego t_z . Niestety narzut odczytu pliku t_o nie jest znany i mocno zależy od buforowania pamięci przez program, system i procesor.

Czas filtracji pojedynczej ramki można rozdzielić dodatkowo na dwie składowe:

$$t_f = t_c + t_p \quad (4.5)$$

Gdzie,

- t_f - czas filtracji pojedynczej ramki,
- t_c - czas przetworzenia ramki,
- t_p - czas porównywania z nastawami filtrów.

W rzeczywistej sytuacji filtrowana ramka będzie już znajdować się w pamięci RAM, dlatego z pomiarów zostanie estymowany czas potrzebny na wykonanie pojedynczego porównania. Zakładając, że czas potrzebny na wykonanie pojedynczego porównania t_j jest stały to czas ten można obliczyć poprzez różnicę czasu wykonania programu z 1 filtrem, a czasem wykonania programu z n -filtrami. Otrzymaną wartość kolejno podzielić przez różnicę liczby filtrów ($n - 1$):

$$t_j = \frac{t_{fn} - t_{f1}}{n - 1} \quad (4.6)$$

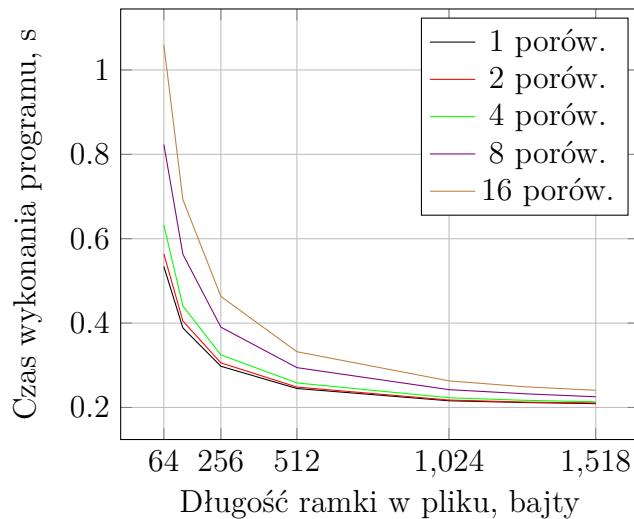
Gdzie,

- t_j - czas pojedynczego porównania,

- t_{f1} - czas filtracji pojedynczej ramki z 1 filtrem,
- t_{fn} - czas filtracji pojedynczej ramki z n-filtrami,

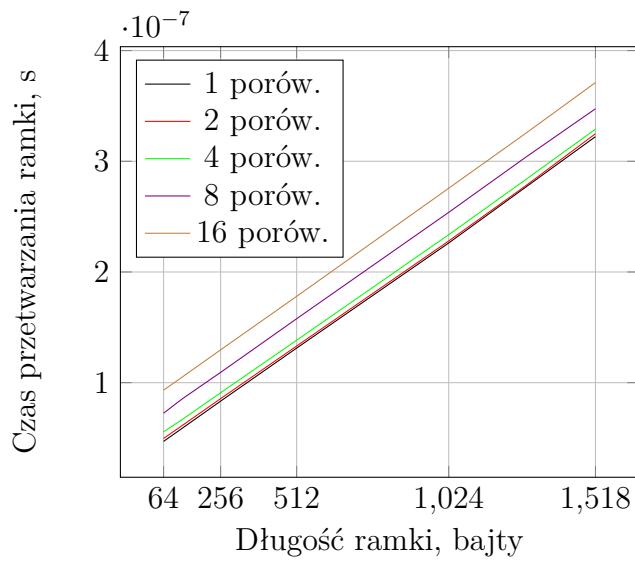
4.3.3 Wyniki

Na rysunku 4.3 przedstawiono uśrednione czasy wykonywania programu *tcpdump* przy filtracji 1 GB pliku dla różnych konfiguracji. Z wykresu można odczytać, że czas rośnie wraz z liczbą nastaw.



Rysunek 4.3: Uśredniony czas wykonywania programu od długości ramki i liczby porównań dla 1GB pliku.

Dzieląc czasy wykonania programu przez liczbę ramek znajdujących się w poszczególnych plikach, otrzymujemy następujący wykres przedstawiony na rysunku 4.4. Można zauważyć, że czas przetwarzania rośnie liniowo w zależności od długości ramki oraz różnica pomiędzy czasami wykonania programu dla zbadanych wariantów liczby porównań jest stała.



Rysunek 4.4: Czas przetwarzania pojedynczej ramki od długości ramki i liczby porównań dla CPU.

Kolejno według wzoru 4.6 został obliczony średni czas pojedynczego porównania. Wyniki dla różnej wielkości ramek znajdują się w tabeli 4.3.

Tablica 4.3: Wyniki pomiarów czasu wykonywania programu.

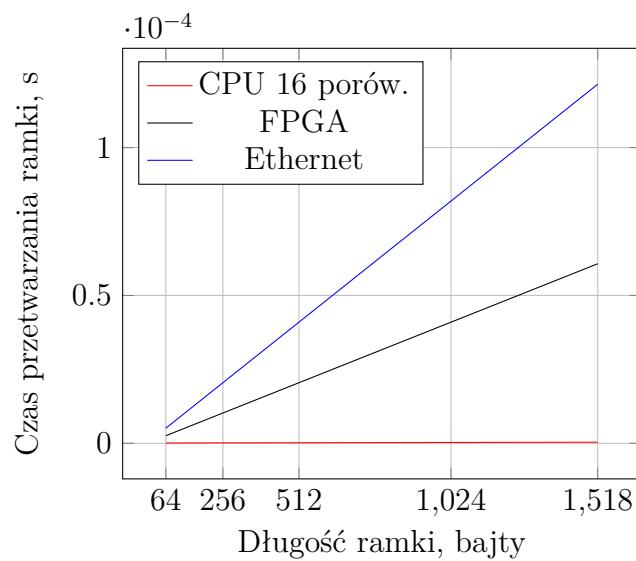
Długość ramki (liczba ramek - k)	Średni czas wykonania \bar{t}_{p1} , s	Średni czas wykonania \bar{t}_{p16} , s	Średni czas pojedynczego porów., $\bar{t}_j = \frac{(\bar{t}_{p16} - \bar{t}_{p1})}{(15 \cdot k)}$, s
64 (11363636)	0.5339	1.0595	3.08E-09
128 (6578947)	0.3885	0.6929	3.08E-09
256 (3571428)	0.2976	0.4631	3.09E-09
512 (1865671)	0.2449	0.3321	3.12E-09
1024 (954198)	0.2158	0.2630	3.30E-09
1280 (766871)	0.2117	0.2490	3.24E-09
1518 (648508)	0.2090	0.2407	3.26E-09

Z wyników przedstawionych w tabeli 4.3 wywnioskować można, że czas jednego

porównania wynosi około 3.08 ns. Wartości dla dłuższych ramek zwierają większy błąd pomiarowy ze względu na mniejszą liczbę ramek do przetworzenia, a co za tym idzie krótszy czas wykonywania pomiaru, dlatego w dalszej części brany pod uwagę będzie czas porównania dla najkrótszej ramki.

Biorąc wartość 3.08 nanosekundy, jako czas pojedynczego porównania to teoretycznie w ciągu 1 sekundy, jednowątkowy program jest w stanie wykonać 325 milionów porównań. Porównując tą wartość z układem FPGA, który w ciągu sekundy w jednym bloku filtracyjnym jest w stanie wykonać 2 miliardy operacji jest wartością sporo mniejszą, jednak są to wartości teoretyczne.

Wykres 4.5 przedstawia rzeczywisty czas odebrania ramki ethernetowej przy prędkości łącza 1 Gbit/s z porównaniem do czasu przetwarzania ramki przez CPU z 16 porównaniami i blokiem filtracyjnym na układzie FPGA. Można zauważyć, że przetwarzania ramki na CPU jest znacznie szybsze. Jest to spowodowane tym, że czas filtracji nie zależy od długości ramki, a głównie od liczby porównań. W przypadku ciągłego napływu danych z 1 Gbit/s sieci może wydawać się, że oba systemy są w stanie przetworzyć ciągły strumień danych, jednak w przypadku CPU może być to założenie błędne.



Rysunek 4.5: Czas przetwarzania pojedynczej ramki od długości ramki i liczby porównań.

Biorąc pod uwagę najbardziej pesymistyczny wariant ruchu sieciowego, czyli odbieranie wyłącznie 64-bajtowych ramek, to czas na przetworzenie i przefiltrowanie ramki dla sieci w standardzie 1000BASE-T wynosi maksymalnie: (64 bajtów ramki + 8 bajtów preambuły + 8 bajtów minimalnego odstępu) * 8 ns = 640 ns. Pojedynczy wątek CPU w takim wypadku jest w stanie wykonać 207 porównań, biorąc jako stały czas 3.08 ns na jedno porównanie. Dla porównania pojedynczy moduł filtrujący niezależnie od długości ramki jest w stanie dla jednej ramki wykonać 224 porównań (28 pól konfiguracyjnych * 2 porównania * 4 filtry).

Można stwierdzić, że moduł filtracyjny zaimplementowany na FPGA jest w stanie odciążyć główny procesor przy filtracji ciągłego strumienia danych. Najbardziej w przypadku bardzo dużego ruchu sieciowego przy niewielkich ramkach i dużej liczby nastaw filtrów. Dodatkowym atutem sprzętowego układu jest jego deterministyczne działanie i 100% poprawność działania niezależna od obciążenia oraz wysoka efektywność energetyczna.

Rozdział 5

Podsumowanie

Cel pracy został osiągnięty. Stworzony system umożliwia monitoring oraz analizę danych na zasadzie filtracji ruchu sieciowego. Podwójny interfejs przechwytywania z precyzyjnymi znacznikami czasowymi daje możliwość wykorzystania systemu do pomiaru opóźnień wprowadzanych przez urządzenia sieciowe działające w standardzie 100/1000BASE-T. Implementacja sprzętowa pozwala na przechwytywanie ruchu praktycznie bez ograniczeń, a sprzętowa filtracja wyraźnie redukuje ilość danych przekazywanych na komputer. Przechwycone dane są zapisane w pliku o standardowym rozszerzeniu *.pcap*, dzięki czemu dalsza analiza danych jest w znacznym stopniu uproszczona. Program wchodzący w skład systemu, wywiera niewielki narzut na pracę systemu, a dodatkowo wielowątkowa struktura programu zapewnia rozłożenie narzutu na dostępne rdzenie.

Przebadano narzut programowego monitoringu ruchu sieciowego i porównano z prezentowanym w pracy systemem. W zakresie analizy porównano filtrację programową ramek ethernetowych na CPU z filtracją wykonywaną na układzie FPGA.

Projekt można rozwijać na wiele sposobów. Przydatną funkcją przy testowaniu urządzeń może być np. możliwość ingerencji w pakiety. Inną ścieżką rozwoju może być opracowanie interfejsu graficznego, który pozwalałby na łatwiejsze ustawianie filtrów i rejestrów sterujących. Ze względu na pominięcie części sieciowej systemu operacyjnego, platforma pozwala na efektywne przetwarzanie danych programowo podczas odbierania danych z karty NetFPGA na komputerze.

Bibliografia

- [1] Pallavi Asrodia, Hemlata Patel. Analysis of various packet sniffing tools for network monitoring and analysis. *International Journal of Electrical, Electronics and Computer Engineering*, 1:55–58, 2012.
- [2] Mateusz Dmitrzak, Kamil Fiedukiewicz, Ireneusz J. Józwik. Analiza metod przetwarzania informacji ruchu sieciowego. *Zeszyty Naukowe. Organizacja i Zarządzanie / Politechnika Śląska*, 32(z. 130):107–115, 2018.
- [3] Sergiy Dorosh, Grzegorz Debita, Patryk Schauer. Network hardware analyzer based on netfpga 1g. *IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2017. doi: 10.1109/WETICE.2017.56.
- [4] Paweł Dymora, Mirosław Mazurek, Dominik Strzałka. Computer network traffic analysis with the use of statistical self-similarity factor. *Annales UMCS Informatica AI*, 13(2):69–81, 2013.
- [5] Excel. White paper: Nvp, what is it for? <https://www.petalit.cz/soubor/excel-whitepaper-nvp/>, [Dostęp 2.11.2019].
- [6] Jyothi G, Pradeep Kumar B, C M Ananda, K C R Nisha. Design and implementation of livetap for deterministic ethernet bus using fpga. *IEEE International Conference On Recent Trends In Electronics Information Communication Technology*, 2016. doi: 10.1109/RTEICT.2016.7807878.
- [7] Paul R. Grams. Ethernet for aerospace applications - ethernet heads for the skies. *Ethernet Technology Summit*, 2015.

- [8] HP. Reduced gigabit media independent interface. https://web.archive.org/web/20160303171328/http://www.hp.com/rnd/pdfs/RGMIIv2_0_final_hp.pdf, [Dostęp 19.10.2019].
- [9] HWiNFO. Oprogramowanie diagnostyczne. <https://www.hwinfo.com/>, [Dostęp 15.10.2019].
- [10] IEEE. 802.3az. <http://www.ieee802.org/3/az/index.html>, [Dostęp 10.10.2019].
- [11] IEEE. 802.3-2018. <https://ieeexplore.ieee.org/servlet/opac?punumber=8457467>, [Dostęp 23.10.2019].
- [12] Ixia. Automotive ethernet: An overview. https://support.ixiacom.com/sites/default/files/resources/whitepaper/ixia-automotive-ethernet-primer-whitepaper_1.pdf, [Dostęp 30.08.2019].
- [13] Microsoft. Visual studio. <https://visualstudio.microsoft.com>, [Dostęp 21.10.2019].
- [14] NetFPGA. <https://netfpga.org/>, [Dostęp 11.10.2019].
- [15] NetSniff-NG. <http://netsniff-ng.org/>, [Dostęp 10.09.2019].
- [16] PCAPNG. file format specification. <https://github.com/pcapng/pcapng>, [Dostęp 9.09.2019].
- [17] PCI-SIG. <https://pcisig.com/faq>, [Dostęp 20.10.2019].
- [18] Thomas B. Preußen, Rainer G. Spallek. Ready pcie data streaming solutions for fpgas. *24th International Conference on Field Programmable Logic and Applications*, 2014. doi: 10.1109/FPL.2014.6927444.
- [19] Python. <https://www.python.org/>, [Dostęp 19.10.2019].
- [20] Justin Rajewski. *Learning FPGAs*. O'Reilly Media, Sebastopol, 2017.

- [21] D Sharath Babu Rao, Dr. V Suma Latha. The evolution of the ethernet various fields of applications. 2015. doi: 10.1109/GET.2015.7453807.
- [22] Scapy. <https://scapy.net/>, [Dostęp 19.10.2019].
- [23] Tcpdump/Libpcap. Link-layer header types. <https://www.tcpdump.org/linktypes.html>, [Dostęp 9.09.2019].
- [24] Tcpdump/Libpcap. Pcap file format specification. <https://www.tcpdump.org/manpages/pcap-savefile.5.txt>, [Dostęp 9.09.2019].
- [25] Malte Vesper, Dirk Koch, Kizheppatt Vipin, Suhaib A. Fahmy. Jetstream: An open-source high-performance pci express 3 streaming library for fpga-to-host and fpga-to-fpga communication. *26th International Conference on Field Programmable Logic and Applications*, 2016. doi: 10.1109/FPL.2016.7577334.
- [26] VITESSE. Gigabit ethernet phy device latency. <https://cdn.microsemi.com/documents/820d3394-b45a-4064-8588-b00246bc224c/download/>, [Dostęp 24.10.2019].
- [27] Wireshark. <https://www.wireshark.org/>, [Dostęp 10.10.2019].
- [28] Xilinx. Axi reference guide, 45–56. https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf, [Dostęp 14.10.2019].
- [29] Xilinx. Axi4-stream infrastructure ip suite v3.0 - logicore ip product guide. https://www.xilinx.com/support/documentation/ip_documentation/axis_infrastructure_ip_suite/v1_1/pg085-axi4stream-infrastructure.pdf, [Dostęp 14.10.2019].
- [30] Xilinx. Clocking wizard v6.0 - logicore ip product guide. https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf, [Dostęp 14.10.2019].

Dodatki

Słownik pojęć

W pracy zostały użyte następujące terminy:

- IP Core - gotowy do wykorzystania blok, którego można użyć przy projektowaniu aplikacji sprzętowej. Blok zazwyczaj jest licencjonowany i może być płatny lub bezpłatny.
- Aplikacja sprzętowa - projekt na układ FPGA napisany w języku opisu sprzętu np. VHDL, Verilog.
- Aplikacja programowa - aplikacja na system operacyjny taki jak Linux lub Windows.
- Tunel - połączenie mostkujące dwa układy PHY, dzięki czemu odbierane dane na jednym układzie PHY są przekazywane na drugi układ PHY i odwrotnie.

Spis skrótów i symboli

TAP - urządzenie pozwalające nasłuchiwać ruch sieciowy na danym połączeniu
(ang. *test access point*)

ASIC – układ scalony zaprojektowany do realizacji z góry określonego zadania.
(ang. *application-specific integrated circuits*)

FPGA - bezpośrednio programowalna macierz bramek (ang. *field-programmable gate array*)

PHY - układ scalony wykorzystywany w warstwie fizycznej modelu ISO/OSI

SDR - sposób przesyłu danych, gdzie stan logiczny sygnałów zatrzaskiwana jest przy opadającym lub narastającym zboczu sygnału zegarowego. (ang. *single data rate*)

DDR - sposób przesyłu danych, gdzie stan logiczny sygnałów zatrzaskiwany jest przy opadającym i narastającym zboczu sygnału zegarowego. (ang. *double data rate*)

Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca w formacie **pdf**, **tex** i **txt**,
- źródła projektu NetCon i skompilowany program,
- projekt sprzętowy w Vivado, testbench i wynikowy bitstream,
- pomiary eksperymentów.
- zbiory danych użyte w analizie czasów filtracji.

Spis rysunków

2.1	Podstawowa ramka Ethernet.	4
2.2	Schemat połączenia urządzenia do sieci Ethernet.	5
2.3	Model ISO/OSI.	8
2.4	Schemat monitoringu z wykorzystaniem regenerującego koncentratora.	9
2.5	Schemat połączenia przełącznika sieciowego.	9
2.6	Schemat monitoringu z użyciem przełącznika oraz funkcji port mirroring'u.	10
2.7	Schemat monitoringu typu man-in-the-middle.	10
2.8	Schemat monitoringu z wykorzystaniem urządzenia typu TAP. . . .	11
2.9	Blokowy model działania pasywnego i aktywnego urządzenia monitorującego.	12
2.10	Zrzut ekranu z programu Wireshark	15
2.11	Zdjęcie karty NETFPGA-1G-CML.	21
2.12	Przykładowy schemat blokowy z programu Vivado.	22
2.13	Schemat blokowy protokołu <i>AXI4-Stream</i>	24
2.14	Przykładowe transakcje z wykorzystaniem protokołu <i>AXI4-Stream</i> . .	25
3.1	Schemat blokowy systemu monitorująco-analizującego.	27
3.2	Schemat blokowy systemu.	28
3.3	Konfiguracja karty NetFPGA z wyłączonymi tunelami.	30
3.4	Konfiguracja karty NetFPGA do pomiaru opóźnień wprowadzanych przez badane urządzenie.	30
3.5	Zależność między długością ramki ethernetowej, a wymaganą przepustowością połączenia PCI-Express przy pełnym nasyceniu łączza. .	31

3.6	Zrzut ekranu z blokowego schematu projektu sprzętowego w programie Vivado.	34
3.7	Uproszczony schemat blokowy pojedynczego modułu.	35
3.8	Schemat blokowy części przechwytyjącej.	36
3.9	Schemat blokowy części pakietyzującej.	38
3.10	Schemat przedstawiający konwersję 8-bitowej szyny do 32-bitowej szyny.	39
3.11	Struktura wyjściowa strumienia danych.	40
3.12	Schemat blokowy części filtracyjnej.	41
3.13	Schemat bloku filtracyjnego.	42
3.14	Rzutowanie odbieranej nastawy na komórkę rejestru konfiguracyjnego.	43
3.15	Struktura rejestrów konfiguracyjnych	43
3.16	Przedstawienie zależności pozycji bajtu w 4-bajtowej części ramki do nastaw.	44
3.17	Pierwszy cykl działania algorytmu filtracji	46
3.18	Drugi cykl działania algorytmu filtracji	47
3.19	Blok implementujący obsługę MDIO.	48
3.20	Przykład przedstawiający działanie mechanizmu manipulacji wartościami rejestru.	49
3.21	Blok kontrolny.	49
3.22	Spersonalizowany blok Xillybus.	51
3.23	Zrzut ekranu z programu Vivado przedstawiający graficzne rozmieszczenie poszczególnych modułów.	52
3.24	Program w trybie monitoringu ruchu sieciowego z zapisem do pliku.	56
3.25	Schemat przedstawiający numerację tuneli.	58
3.26	Zrzut ekranu programu po wywołaniu funkcji mdio.	59
3.27	Wygląd stacji roboczej z włożoną kartą NetFPGA-1G-CML.	60
3.28	Wygląd włożonej karty NetFPGA-1G-CML z podłączonym programatorem z bliska.	60
3.29	Zrzut ekranu z programu Vivado przed wgraniem tzw. bitstream'a.	61
3.30	Zrzut ekranu z okna menedżera urządzeń.	62
3.31	Konfiguracja pozwalająca na pomiar opóźnień tunelu.	62

3.32 Zdjęcie przedstawiające podłączenie w konfiguracji umożliwiającej pomiar opóźnień tunelu. (Na dole karta NetFPGA, nad nią karta sieciowa).	63
3.33 Struktura nazewnictwa zapisywanych plików.	63
3.34 Zrzut ekranu z programów w trybie przechwytywania i zapisywania ruchu sieciowego do pliku.	64
3.35 Zrzut ekranu z programu po wygenerowaniu ramek.	65
3.36 Zrzut ekranu po otwarciu pliku przechwytywania w programie Wireshark.	66
3.37 Konfiguracja pozwalająca na pomiar opóźnień.	67
4.1 Konfiguracja pozwalająca na pomiar opóźnień.	72
4.2 Histogram opóźnień w zależności od długości przewodu.	74
4.3 Uśredniony czas wykonywania programu od długości ramki i liczby porównań dla 1GB pliku.	80
4.4 Czas przetwarzania pojedynczej ramki od długości ramki i liczby porównań dla CPU.	81
4.5 Czas przetwarzania pojedynczej ramki od długości ramki i liczby porównań.	82

Spis tablic

2.1	Struktura nagłówka formatu pliku PCAP.	16
2.2	Przepustowość PCI Express [17].	19
3.1	Tabela przedstawiająca wykorzystanie dostępnych komórek i modułów układu FPGA.	53
4.1	Wyniki pomiarów zdolności przechwytywania przy wysokim obciążeniu łącza.	71
4.2	Wyniki eksperymentu pomiaru czasu propagacji.	75
4.3	Wyniki pomiarów czasu wykonywania programu.	81