GEM External FIFO

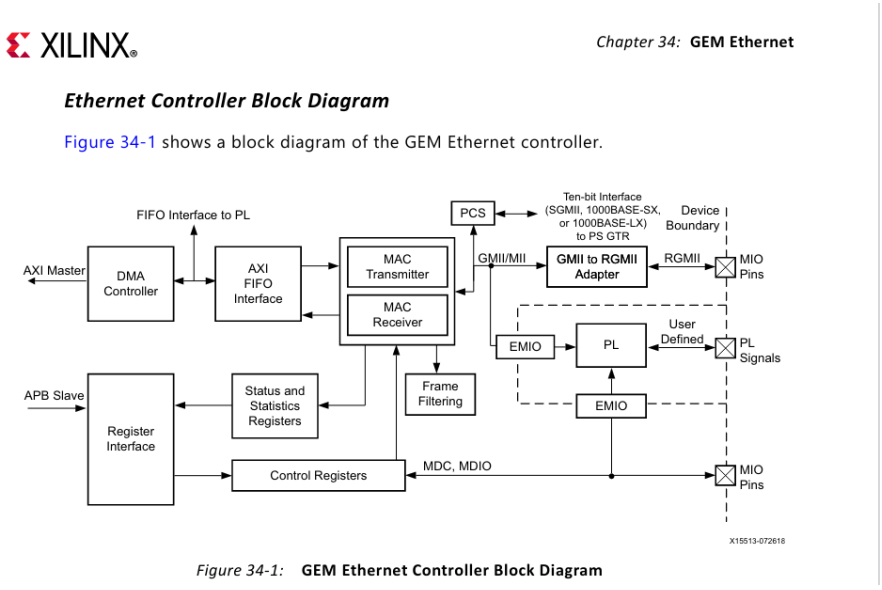
Michał Bojke

Sierpień 2022

1 Spis treści

2 Cel użycia

Interfejs External FIFO Interface jest wykorzystany w celu odciążenia procesora poprzez przekierowanie pakietów odbieranych przez GEM do PL, omijając przy tym blok DMA zintegrowany w GEMie. Przetwarzanie pakietów w PL powinno zwiększyć przepływność, a co za tym idzie - rozwiązać problem przesyłania małych ramek.



Rys. 1: Diagram blokowy interfejsu GEM. Interesująca nas część to FIFO interface to PL, w lewym górnym rogu ilustracji.

3 Ustawienie platformy testowej

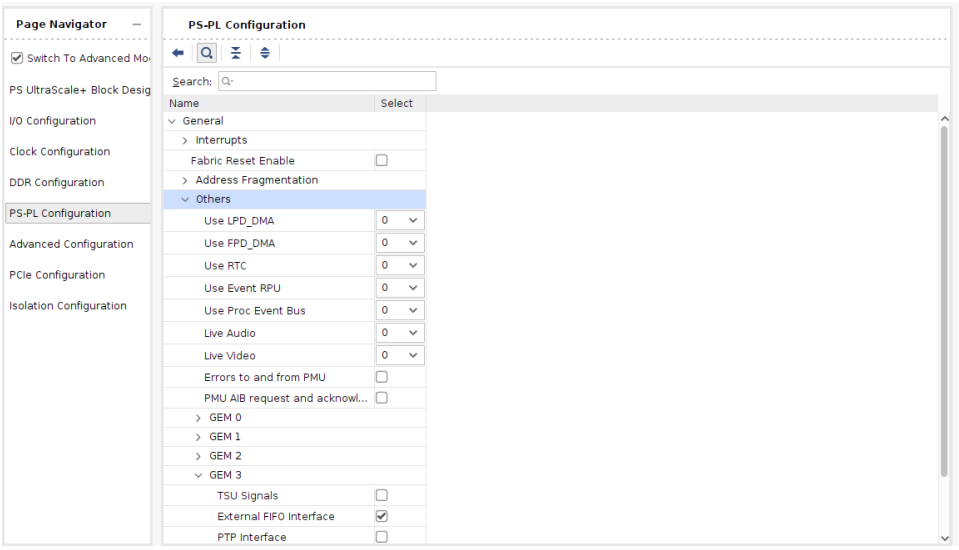
3.1 Ustawienia procesora z użyciem Vivado

Konfiguracja procesora obywa się zgodnie z opisami zamieszczonymi w: `https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/478937213/MPSoC+PS+and+PL+Ethernet+Example+Projects#Using-PL-1G-Ethernet`

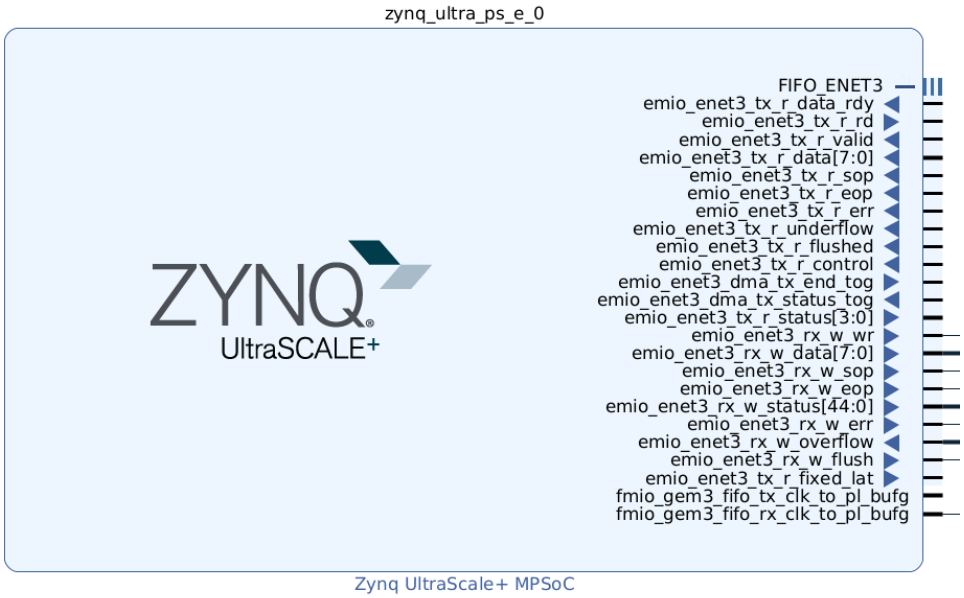
Łatwo jest skonfigurować procesor opierając się na przykładach z: `https://github.com/Xilinx-Wiki-Projects/ZCU102-Ethernet`

Konkretnie z użyciem projektu opisanego jako ps_mio_eth_1g. W skrypcie do generacji projektu w Vivado (Scripts/project_bd.tcl) należy zmienić wersję programu na tą, której używamy.

Jedyną modyfikacją poza podstawowym włączeniem interfejsu GEM3 na tym etapie jest ustawienie External FIFO Interface w ustawieniach procesora (zakładka PS-PL Interface/General/Others/GEM3/External FIFO interface):



Rys. 2: Włączenie External FIFO Interface



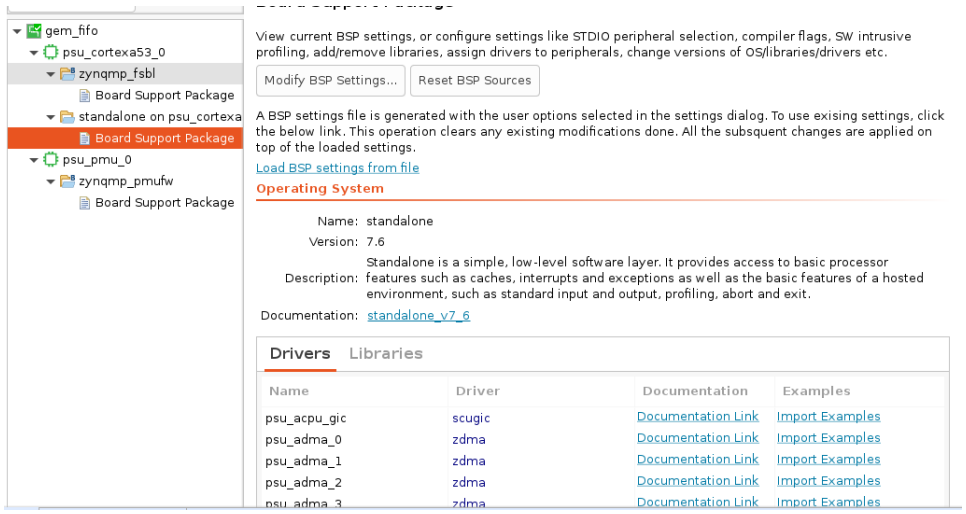
Rys. 3: Blok procesora po konfiguracji

Sygnały opisane są w:
`https://docs.xilinx.com/v/u/en-US/ug1085-zynq-ultrascale-trm`
(GEM Ethernet/Functional Description/Rx and Tx FIFO Interfaces to PL)

Należy zwrócić uwagę, że szerokość rejestru rx_w_data wynosi 8 bitów, w przeciwieństwie do 32 bitów podanych w dokumentacji – dzieje się tak, ponieważ używamy external FIFO skonfigurowanego jako slave. Aby rozpocząć pracę w Vitis, należy wyeksportować projekt w Vivado (File/Export Hardware), po dokonaniu implementacji i generacji bitstreamu.

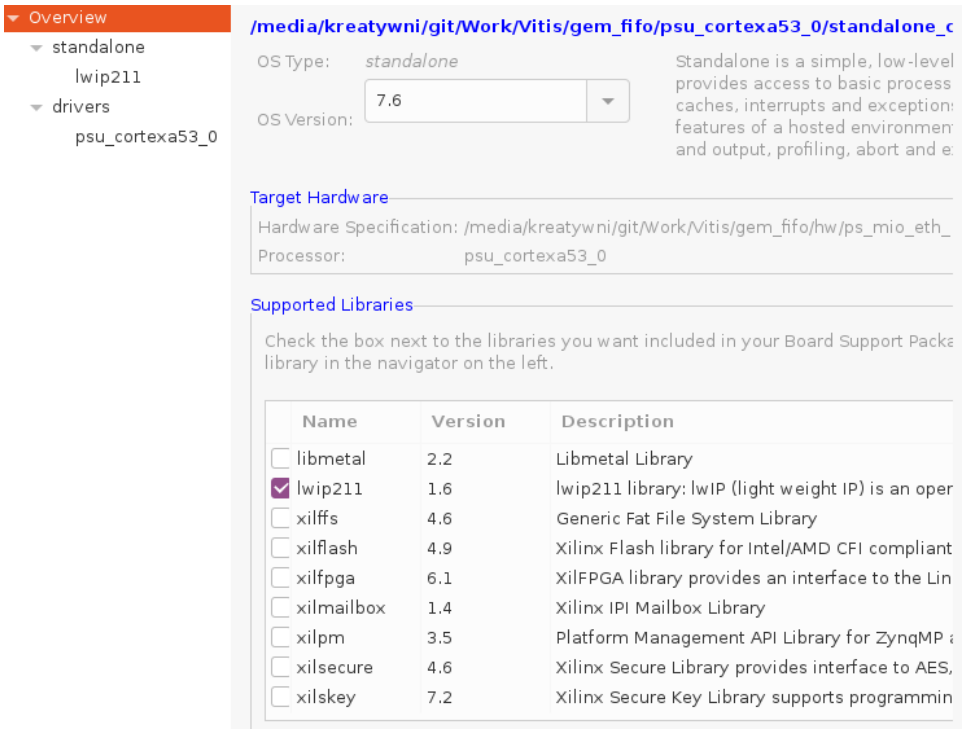
3.2 Aplikacja testowa z użyciem lwip

Do przygotowania testu, należy stworzyć platformę w Vitis (File/New/Platform Project). Następnie, przez plik platform.spr, należy zmodyfikować ustawienia BSP platformy standalone:



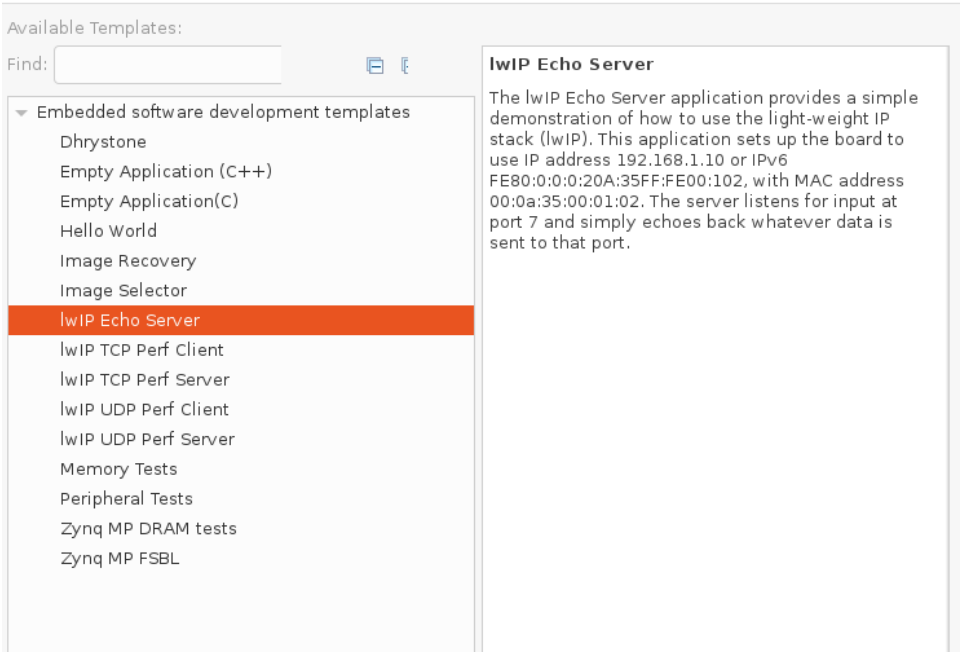
Rys. 4: Modyfikacja ustawień BSP platformy standalone

W menu Modify BSP Settings, załączamy bibliotekę lwip:



Rys. 5: Załączenie biblioteki lwip

Uwaga: W opcjach lwip możemy włączyć debugowanie poszczególnych elementów aplikacji, co jest dość przydatne w zrozumieniu jak działa lwip. Następnie budujemy projekt oraz tworzymy nową aplikację (File/New/Application Project). Z szablonu należy wybrać projekt lwip, np.. Echo Server:



Rys. 6: Stworzenie projektu lwip z szablonu.

W wygenerowanym szablonie w pliku main.c należy ustawić IP, maskę oraz bramę.

```
#else
    /* initialize IP addresses to be used */
    IP4_ADDR(&ipaddr,  10, 60,   22, 10);
    IP4_ADDR(&netmask, 255, 255, 0,  0);
    IP4_ADDR(&gw,      10, 60,   255,  254);
#endif
```

Rys. 7: Zmiana adresów IP w main.c

Istnieje również możliwość nadania płytce adresu MAC. Domyślny adres, nadawany przez aplikację, to 02:01:00:35:0A:00. Po zbudowaniu aplikacji, powinna ona ustawiać działający serwer echo. Uwagi: Na tym etapie, nasza aplikacja działa bez wykorzystania External FIFO Interface, ale możemy go włączyć z jej poziomu, o czym mowa w punkcie 3. Włączenie interfejsu sprawia, że aplikacja nie działa poprawnie – lwip korzysta z DMA GEMu, co oznacza, że serwer nigdy nie wykrywa przychodzących pakietów (ponieważ nie trafiają one nigdy do DMA GEMu, lecz wysyłane są do PL). Więcej o tym problemie w punkcie 7.

4 Opis i konfiguracja External FIFO Interface

4.1 Opis działania

Użycie interfejsu External FIFO pozwala przesłać dane bezpośrednio do PL, dzięki czemu nie musimy polegać na kontrolerze DMA GEM. Interfejs odbiera ramki w następujących przypadkach (wszystkie przytoczone rejestry opisane są w podpunkcie 4.3):

1. Adres MAC źródła/odbiorcy ramki jest taki, jak skonfigurowano w rejestrach spec_add[1...4]_top oraz spec_add[1...4]_bottom. Przy inicjalizacji sterownik zapisuje do pierwszego takiego rejestru adres MAC urządzenia, który ustawiony jest w projekcie.

2. Typ/dlugość ramki przychodzącej jest zgodna z ID zapisanym w którymś z rejestrów spec_type[1...4]

3. Jeśli włączone jest hashowanie unicast/multicast poprzez rejestr network_config, to przychodząca ramka jest przyjęta tylko w przypadku poprawnego rozpoznania.

4. Adres odbiorcy ramki to 0xFFFFFFFFFFFF i broadcasty są włączone (rejestr network_config).

5. Włączone jest odbieranie wszelkich ramek (rejestr network_config).

Uwagi: Gdy wpiszemy w rejestry spec_add[1...4] adresy urządzenia-nadawcy, odbierane będą wszystkie pakiety wysyłane przez to urządzenie - również broadcast, nawet jeśli są wyłączone.

4.2 Włączenie interfejsu oraz edytowanie zawartości rejestrów

Aby łatwiej konfigurować rejestry, należy załączyć do projektu plik nagłówkowy "xemacps.h", tj. Plik nagłówkowy sterownika Xilinx Embedded Processor Block Ethernet. Zawiera on definicje większości potrzebnych offsetów rejestrów dotyczących interfejsu GEM. Najważniejsze z nich zdefiniowane są w pliku "xemacps_hw.h". Niestety, w pliku nie jest zdefiniowany offset, który potrzebny jest najbardziej – external_fifo_interface. Dlatego też należy zdefiniować go własnoręcznie:

```
#define XEMACPS_JUMBOMAXLEN_OFFSET    0x00000048U /**< Jumbo max length reg */
#define XEMACPS_EXTFIFOIF_OFFSET      0x0000004CU /**< Enable FIFO reg */
#define XEMACPS_RXWATERMARK_OFFSET    0x0000007CU /**< RX watermark reg */
```

Rys. 8: Definicja XEMACPS_EXTFIFOIF_OFFSET.

Żeby wpisywać wartości do rejestrów, należy użyć funkcji XEmacPs_WriteReg, która jako argumenty przyjmuje bazowy adres rejestru, offset, oraz wartość, którą do niego wpisuje.

```
XEmacPs_WriteReg(PLATFORM_EMAC_BASEADDR, XEMACPS_EXTFIFOIF_OFFSET, 1);
```

Rys. 9: Przykład użycia funkcji XEmacPs_WriteReg ze zdefiniowanym wcześniej offsetem.

Powyższy przykład włącza External FIFO Interface. Opis wszystkich rejestrów można znaleźć tu (sekcja GEM Module):
`https://www.xilinx.com/htmldocs/registers/ug1087/ug1087-zynq-ultrascale-registers.html`

4.3 Rejestry niezbędne

Przy inicjalizacji sterownik ustawia większość rejestrów niezbędnych do prawidłowego funkcjonowania interfejsu. W przypadku, gdy chcemy jednak pracować z External FIFO, niezbędna jest dodatkowa konfiguracja rejestru network_config.

1. external_fifo_interface (offset 0x000000004C)

Nazwa	Nr.bitu	Opis
–	31:1	Ignorowane
external_fifo_interface	0	Ustawiony na 1, włącza External FIFO.

2. network_config (offset 0x0000000004)

Nazwa	Nr.bitu	Opis
data_bus_width	22:21	Aby pracować w trybie External FIFO, muszą być wyzerowane.

4.4 Rejestry dodatkowe

Poniżej znajduje się lista przykładowych przydatnych rejestrów i ich poszczególnych bitów, które pozwalają dowolnie modyfikować sposób działania interfejsu.

1. network_config (offset 0x0000000004)

Nazwa	Nr.bitu	Opis
receive_checksum _offload_enable	24	Gdy ustawiony, obliczana jest suma kontrolna przychodzących pakietów. Pakiety o niewłaściwej sumie są odrzucane.
data_bus_width	22:21	Aby pracować w trybie External FIFO, muszą być wyzerowane.
no_broadcast	5	Gdy ustawiony, ramki broadcast zaadresowane na FFFFFFFF nie będą przyjmowane.
copy_all_frames	4	Gdy ustawiony, wszystkie poprawne ramki będą przyjmowane.
jumbo_frames	3	Gdy ustawiony, włącza obsługę ramek jumbo o dopuszczonej długości ustawionej przez rejestr jumbo_max_length. Domyślnie 10 240 bajtów.
discard_non _vlan_frames	2	Gdy ustawiony, przepuszczone zostaną tylko ramki VLAN.

2. jumbo_max_length (offset 0x0000000048)

Nazwa	Nr.bitu	Opis
—	31:16	Ignorowane
jumbo_max_length	15:0	Ustawia maksymalną długość ramek jumbo.

3. spec_add[1...4]_bottom (offsety 0x0000000088; 0x0000000090; 0x0000000098; 0x00000000A0)

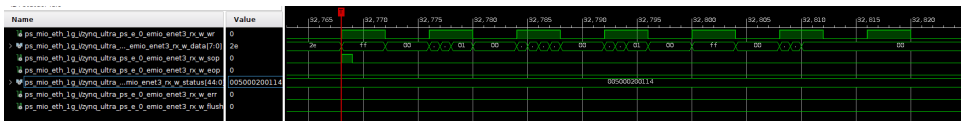
Nazwa	Nr.bitu	Opis
address	31:0	Mniej znaczące bity adresu MAC nadawcy/odbiorcy, w zależności od ustawienia spec_add[1...4]_top. Bit 0 określa multicast/unicast.

4. spec_add[1...4]_top (offsety 0x000000008C; 0x0000000094; 0x000000009C; 0x00000000A4)

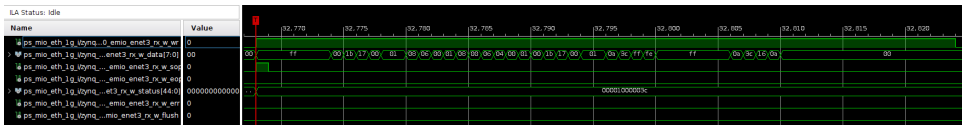
Nazwa	Nr.bitu	Opis
—	31:17	Ignorowane
filter_type	16	Bit decydujący o tym, czy pakiety przyjmowane są na podstawie MAC nadawcy, czy odbiorcy. Gdy jest wyzerowany, liczy się MAC odbiorcy, a gdy ustawiony, liczy się adres MAC nadawcy, z którego pochodzi ramka.
address	15:0	Bardziej znaczące bity adresu MAC nadawcy/odbiorcy, w zależności od ustawienia filter_type.

4.5 Działanie interfejsu

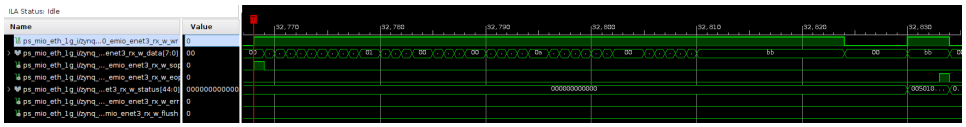
Poniżej zamieszczone zostały przebiegi czasowe, ilustrujące odbiór danych w różnych konfiguracjach.



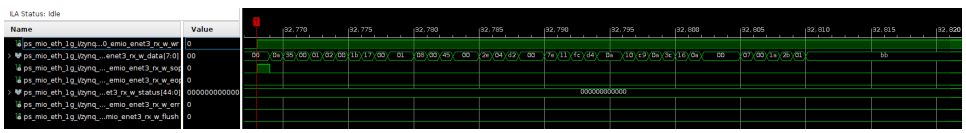
Rys. 10: Przebieg czasowy dla włączonego External FIFO. Do rejestru external_fifo_interface wpisana została wartość 1.



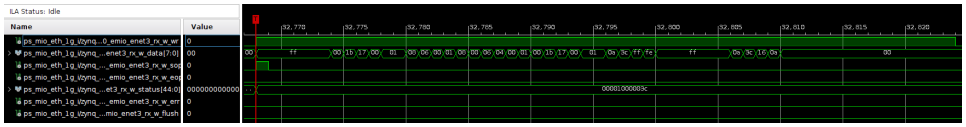
Rys. 11: Przebieg czasowy z rys. 10. Do rejestru network_config wpisana została wartość 0x011F24C2 (Data Bus Width dostosowany do pracy z External FIFO). Jak widać, wartości wpisywane są poprawnie – w sygnale rx_w_wr nie ma przerw.



Rys. 12: Przebieg czasowy z rys. 11. Do rejestru network_config wpisana została wartość 0x011F24E2 (Wyłączone odbieranie broadcastów). Odebrana została wysłana w ramach testu ramka o treści "BB BB BB BB".



Rys. 13: Przebieg czasowy z rys. 12. Do rejestru spec_add1_high wpisana została wartość 0x000000201, a do rejestru spec_add1_bottom wartość 0x00350A00. Oznacza to, że ramki odbierane filtrowane są na podstawie adresu MAC odbiorcy - w tym przypadku MAC płytki ZCU102, który został jej nadany w aplikacji.



Rys. 14: Przebieg czasowy z rys. 13. Do rejestru spec_add1_high wpisana została wartość 0x000010101, a do rejestru spec_add1_bottom wartość 0x00171B00. Oznacza to, że ramki odbierane filtrowane są na podstawie adresu MAC nadawcy. Warto zauważyć, że mimo wyłączenia odbierania broadcastów ramka została odebrana, ponieważ adres MAC nadawcy został ustawiony jako filtr.

5 Testy iperf UDP/TCP bez External FIFO Interface

Testy zostały wykonane z użyciem przykładowych aplikacji lwip UDP/TCP perf server, na platformie skonfigurowanej tak jak w punkcie 2 (z wyłączonym External FIFO Interface). Maksymalna bezstratna przepływność w testach UDP jest definiowana jako utrata pakietów na poziomie mniejszym bądź równym 1%.

5.1 Serwer UDP

Rozmiar ramki	Maksymalna bezstratna przepływność [Mb/s]
64	99.4
128	211
256	302
1024	937
1518	937

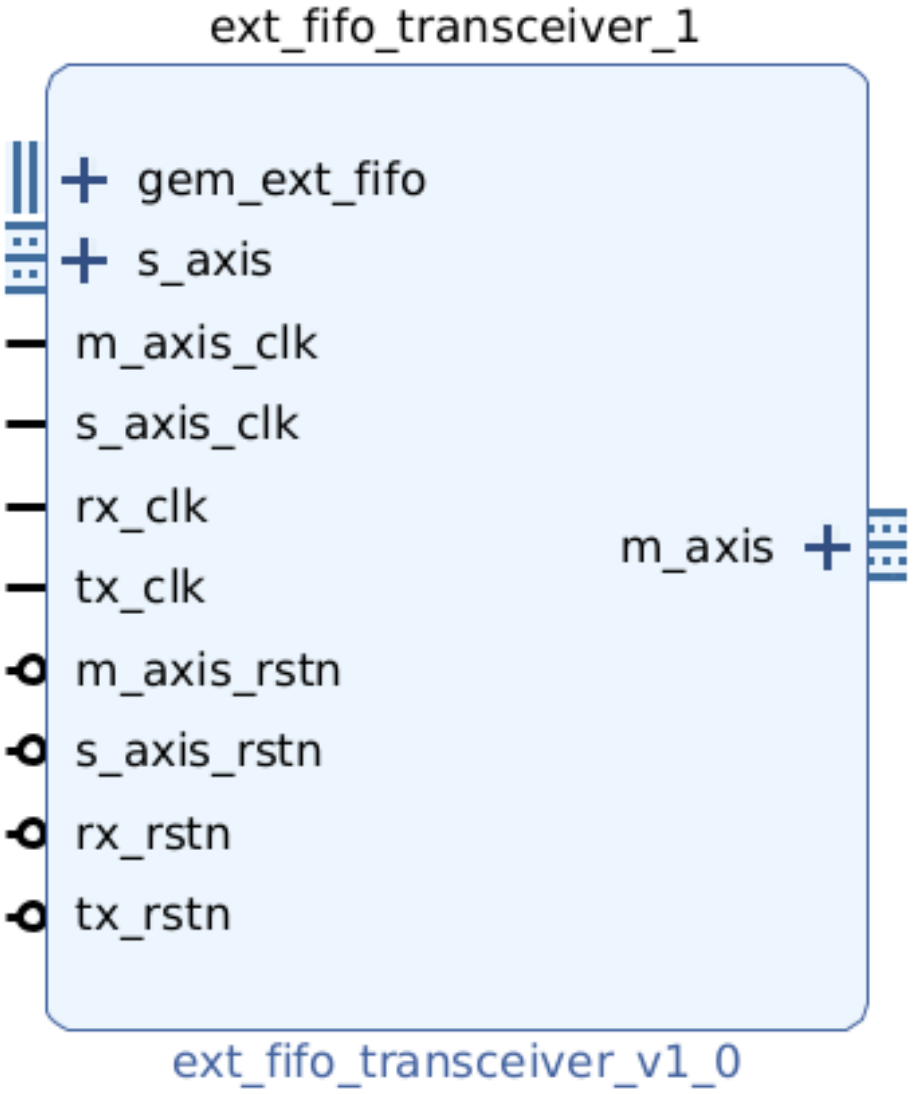
5.2 Serwer TCP

Rozmiar ramki	Przepływność [Mb/s]
64	96.1
128	95.9
256	95.6
1024	95.2
1518	95.4

6 Opis bloku IP

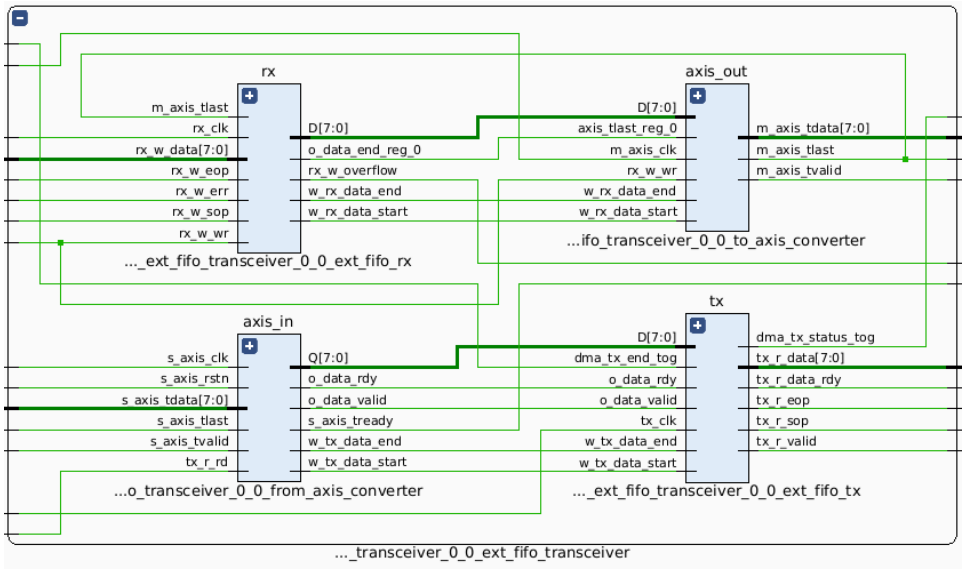
6.1 Ogólny opis

Aby poprawnie odebrać pakiety ethernetowe od GEM zaprojektowany został poniższy blok IP:



Rys. 15: Blok IP ext_fifo_transceiver.

Składa się on z 4 modułów: odbiornika, konwertera do-AXI-Stream, konwertera z-AXI-Stream, oraz nadajnika:



Rys. 16: Blok IP ext_fifo_transceiver podzielony na moduły.

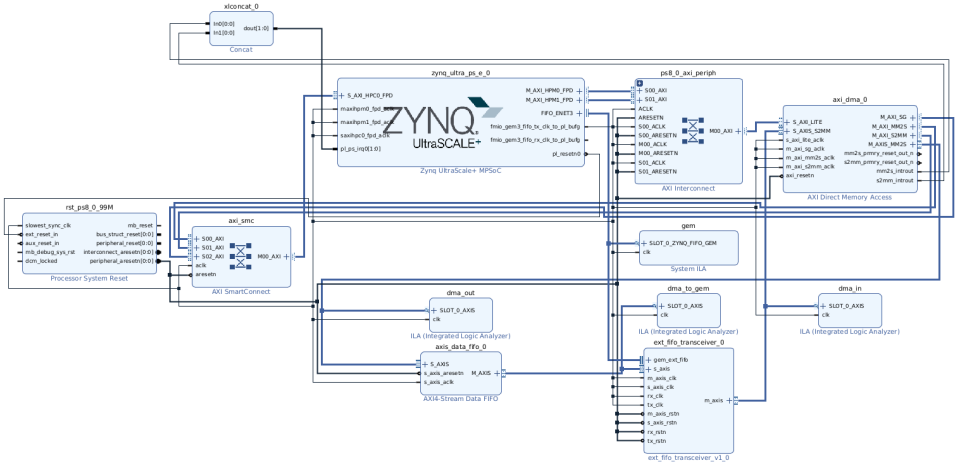
6.2 AXI-Stream

Blok IP został skonfigurowany tak, aby można było podłączyć go do External FIFO Interface (połączenie gem_ext_fifo). Odebrane przez to połączenie dane przesyłane są przez wyjście m_axis, natomiast przychodzące przez wejście s_axis dane wysyłane są z powrotem do GEM. Dzięki temu, blok można połączyć z dowolnymi blokami działającymi z AXI-Stream (w szczególności z blokiem AXI-Stream-FIFO oraz kontrolerem AXI-DMA). Ext_fifo_transceiver nie używa następujących sygnałów AXI-Stream: STRB, TDEST, TKEEP, TID, TUSER. Wynika to z braku takiej potrzeby; implementacja tych sygnałów nie jest skomplikowana, więc w razie potrzeby jest to możliwe. Uwagi: Ext_fifo_transceiver nie implementuje kolejki FIFO, dlatego polega na podłączeniu bloku AXI-Stream-FIFO na wyjściu i/lub wejściu.

7 Konfiguracja platformy testowej z użyciem AXI DMA

7.1 Block design z użyciem Vivado

Aby przetestować sposób, w który External FIFO Interface współpracuje z kontrolerem DMA umieszczonym w PL, utworzona została poniższa platforma testowa:



Rys. 17: Platforma testowa, wykorzystująca External FIFO Interface, stworzony blok IP, AXI-Stream-FIFO oraz kontroler AXI-DMA.

Blok AXI-DMA został skonfigurowany tak, aby działał w trybie Scatter-Gather. Wszystkie bloki AXI-Stream działają na 8-bitowych danych.

7.2 Opis działania aplikacji testowej z użyciem Vitis

Aplikacja testowa została stworzona na podstawie przykładu lwip Echo Server. Działa ona następująco:

1. Pakiety odebrane przez GEM przesyłane są do bloku IP
2. Blok IP konwertuje dane na AXI-Stream i przesyła je do AXI-DMA
3. AXI-DMA, z użyciem przerwania przesyła dane do procesora, który od razu przesyła dane z powrotem do AXI-DMA
4. AXI-DMA wysyła dane odebrane od procesora do AXI-Stream-FIFO

5. AXI-Stream-FIFO wysyła dane do bloku IP

6. Blok IP odbiera dane z AXI-Stream, a następnie wysyła je do GEM.

Jest to oczywiście zwyczajny loopback, ilustruje on jednak jak można wykorzystać PL do obróbki/przesyłu odebranych ramek – wystarczy dodać blok obrabiający dane pomiędzy ext_fifo_transceiver a AXI-DMA, lub obrabiać dane po przesłaniu ich z AXI-DMA do procesora, a potem przesyłać je z powrotem do PL bez udziału procesora.

7.3 Opis aplikacji testowej

Przykładowa wersja aplikacji opiera się na zmodyfikowanej przykładowej aplikacji lwip Echo Server.

```
while (1)
{

    /* When the RxCallback sets RxDone flag, immediately send the packet forward */
    if (RxDone != 0)
    {
        /* Send a packet */
        Status = SendPacket(&AxiDma);
        if (Status != XST_SUCCESS)
        {
            xil_printf("Failed send packet\r\n");
            return XST_FAILURE;
        }
        RxDone = 0;
    }

    if (TcpFastTmrFlag)
    {
        tcp_fasttmr();
        TcpFastTmrFlag = 0;
    }
    /* This detects whether a pbuf is present, and starts a PCB connection if it is */
    if (TcpSlowTmrFlag)
    {
        tcp_slowtmr();
        TcpSlowTmrFlag = 0;
    }
    xemacif_input(echo_netif);
    transfer_data();
}

/* never reached */
cleanup_platform();
```

Rys. 18: Główna pętla programu.

7.4 Działanie DMA

Inicjalizacja DMA oraz funkcje wspomagające zostały odwzorowane z poniższego źródła:
`https://github.com/Xilinx/embeddedsw/blob/b9b64f53e11723c8df0dfda1c59742428b6f1df1/XilinxProcessorIPLib/drivers/AXIdma/examples/xAXIdma_example_sg_intr.c`
Kod inicjalizujący kontroler DMA oraz funkcje pomocnicze zostały umieszczone w pliku dma_handler.c . Funkcja zapisująca przychodzące pakiety (RxCallback) została zmodyfikowana tak, aby pakiety zostały nadpisywane w jednym miejscu w pamięci, natomiast funkcja wysyłająca pakiety (SendPacket) została zmodyfikowana tak, aby pobierać pakiety z tego samego adresu pamięci a następnie je wysyłać. Wywoływana jest z głównej pętli programu po ustawieniu flagi RxDone przez przerwanie.

7.5 Działanie LWIP

Lwip działa z użyciem przerwań wywyoływanych w regularnych odstępach, które ustawiają flagi TcpFastTmr oraz TcpSlowTmr. Wywoływane przez flagi funkcje sprawdzają, czy aktywny jest obecnie PCB (Protocol Control Block), który zarządza sesją TCP. Odebrane przez procesor pakiety lwip alokuje z wykorzystaniem struktury pbuf (od packet buffer), która zostaje zakolejkowana w strukturze netif, która odpowiedzialna jest za całą komunikację internetową. Sprawa komplikuje się, gdyż aplikacja łączy inicjalizację netif z inicjalizacją sterownika EMacPs, odpowiedzialnego za "wyciąganie" pakietów z hardware'u.

```
/* Add network interface to the netif_list, and set it as default.
 * This also initalizes the EmacPs driver with nwcfg register values, which we overwrite later.. */
if (!xemac_add(echo_netif, &ipaddr, &netmask,
                    &gw, mac_ethernet_address,
                    PLATFORM_EMAC_BASEADDR)) {
    xil_printf("Error adding N/W interface\n\r");
    return -1;
}
```

Rys. 19: Xilinxowa funkcja dodająca netif do lwip.

Funkcja xemac_add wywołuje następnie pochodzącą z lwip funkcję netif_add, a także inicjalizuje interfejs xemacpsif. Ta z kolei wywołuje funkcję low_level_init, gdzie, w strukturze xemacpsif_s, alokowana jest pamięć na kolejkę przychodzących oraz wychodzących pbufów. Struktura xemacpsif_s zawiera również w sobie strukturę xemacps, która z kolei zawiera informacje o DMA GEMa (między innymi adresy, w których zostają alokowane Buffer Descriptor Rings). Inicjalizacja DMA znajduje się w pliku xemacpsif_dma.c .

7.6 Ustawienie przerwań

Aby przerwania lwip oraz AXIDMA działały poprawnie, postanowiłem umieścić je w pliku platform_zynqmp.c , w funkcji platform_setup_interrupts oraz platform_enable_interrupts, a także ustawić ich priorytety w następujący sposób:

```
XScuGic_SetPriorityTriggerType(IntcInstancePtr, TX_INTR_ID, 0xA0, 0x3);
XScuGic_SetPriorityTriggerType(IntcInstancePtr, RX_INTR_ID, 0xA0, 0x3);
XScuGic_SetPriorityTriggerType(IntcInstancePtr, TIMER_IRPT_INTR, 0xA8, 0x3);
```

Rys. 20: Ustawienia priorytetów przerwań (trzeci argument funkcji, wyższy numer oznacza niższy priorytet).

8 Problemy do rozwiązania

8.1 DMA i lwip

Aplikacja konfiguruje lwip tak, aby korzystało z DMA GEMa, co sprawia, że nie odpowiada ona na pakiety przychodzące do procesora z innego kontrolera DMA. Przez to, testowanie przepływności External FIFO Interface jest niemożliwe z wykorzystaniem przykładowej wersji aplikacji UPD/TCP Perf Server.

8.2 Rozpoczęcie sesji TCP

Z racji tego, że lwip nie wykrywa pakietów odbieranych, nie jest w stanie również ustanowić sesji TCP. Sprawia to, że platforma jest w stanie jedynie odbierać początkowy pakiet TCP, lecz nie potrafi przesłać go dalej (ponieważ nigdy nie sygnalizuje, że jest gotowa na odebranie kolejnych pakietów). W związku z tym, obecnie przesyłanie działa tylko na pakietach UDP i innych, niewymagających sesji (np. ICMP).

8.3 Potencjalne rozwiązania:

- Skonfigurować lwip tak, aby używało AXI-DMA zamiast DMA GEMa w całości.
- Napisanie własnych pbufów, które odwołują się do buffer descriptorów z AXI-DMA, zgodnie z przykładem:
  `https://www.nongnu.org/lwip/2_1_x/zerocopyrx.html`

- Zmiana przerwań AXI-DMA tak, aby po odbiorze pakiety alokować pbuf w strukturze xemacps. W teorii, wykrycie pbufu w tej strukturze powinno aktywować PCB TCP, przez co (jeśli zawartość odebranego pakietu jest odpowiednia) rozpoczęta zostanie sesja TCP.
- Stworzenie własnego network interface, dodanie go do listy aktywnych netifów lwip (zamiast użycia funkcji xemac_add używamy wtedy normalnego netif_add). Należy wtedy jednak zainicjalizować osobno XEmacPs, aby "wyciągał" pakiety z hardware'u. Pomocny link: `https://lwip.fandom.com/wiki/Writing_a_device_driver`
- Przetestowanie przepływności bez użycia lwip, a jedynie iperf ustawionego na komputerze jako serwer, a platformą testową jako client.

9 Przydatne narzędzia

- Ostinato - służy do generacji pakietów TCP/UDP, można ustawić długość, zawartość, ilość pakietów etc. Niestety nie jest w stanie uczestniczyć w sesji TCP.
- Telnet - używany tylko do badania sesji TCP.
- CocoTB – do pisania testbenchów w Pythonie, aby przetestować działanie napisanego w Verilogu bloku IP. Przykładowe testy znajdują się w repozytorium.

10 Podsumowanie

Sam mechanizm obsługi External FIFO Interface w konfiguracji z AXIDMA działa bez zarzutu, jednak bez zmian w bibliotece lwip bądź sterowniku EmacPs niemożliwym jest na ten moment przetestowanie zysku przepływności.