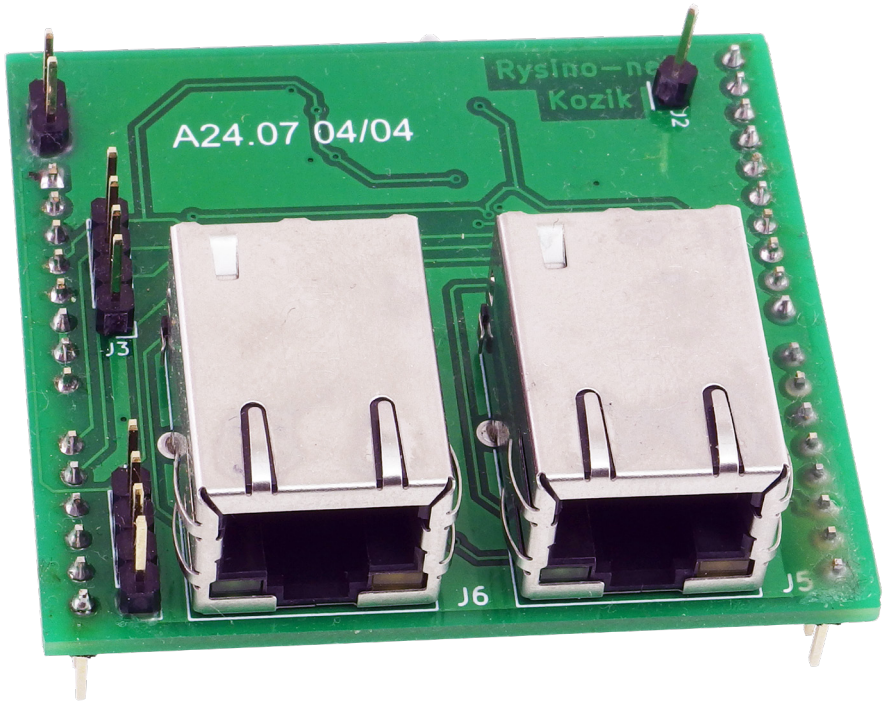
Shield z Ethernetem

do płytki uruchomieniowej Rysino z układem FPGA Intel Max10

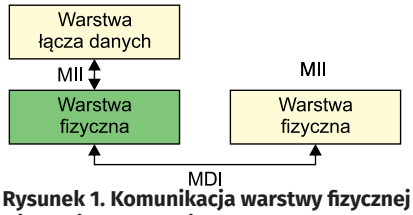
Obecnie chyba najpopularniejszym standardem sieci jest Ethernet. Definiuje on dwie najniższe warstwy stosu sieciowego: fizyczną i łącza danych. Prezentowany projekt realizuje funkcjonalności warstwy pierwszej dla wersji standardu pozwalającej na przesył danych z prędkością 100 Mb/s. Jest to tak zwany Fast Ethernet.



Moduł został zaprojektowany do współpracy z płytką Rysino wyposażoną w układ FPGA Intel Max10. Zawiera warstwę fizyczną dla dwóch portów Ethernet 100 Mb/s. Warstwa fizyczna, tak jak pokazuje to **rysunek 1**, komunikuje się z jednej strony z warstwą łącza danych, a z drugiej z warstwą fizyczną w innym urządzeniu. Komunikacja z wyższą warstwą odbywa się przez jeden z tak zwanych interfejsów niezależnych od medium (*media-independent interface*). Istnieje kilka standardów różniących się szerokością linii danych. My wykorzystamy RMII, czyli *reduce media-independent interface* – zredukowany interfejs niezależny od medium. Składa się on z siedmiu linii:

- RxD[1:0] – dwie linie odbieranych danych,
- CRS_DV – połączony sygnał poprawności odebranych danych i informacji o zajętości łącza,
- TxD[1:0] – dwie linie wysyłanych danych,
- TX_EN – sygnał poprawności nadawanych danych,
- CLK – wspólny zegar o częstotliwości 50 MHz.

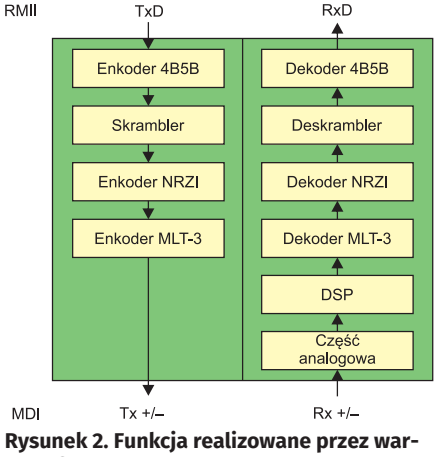
Do połączenia z innymi urządzeniami używany jest tak zwany interfejs zależny od medium (*media-dependent interface*), w skrócie MDI. Zwykle używa się tutaj 8-żyłowej skrętki oraz złączy RJ-45. Jednak wykorzystywane są tylko cztery linie:



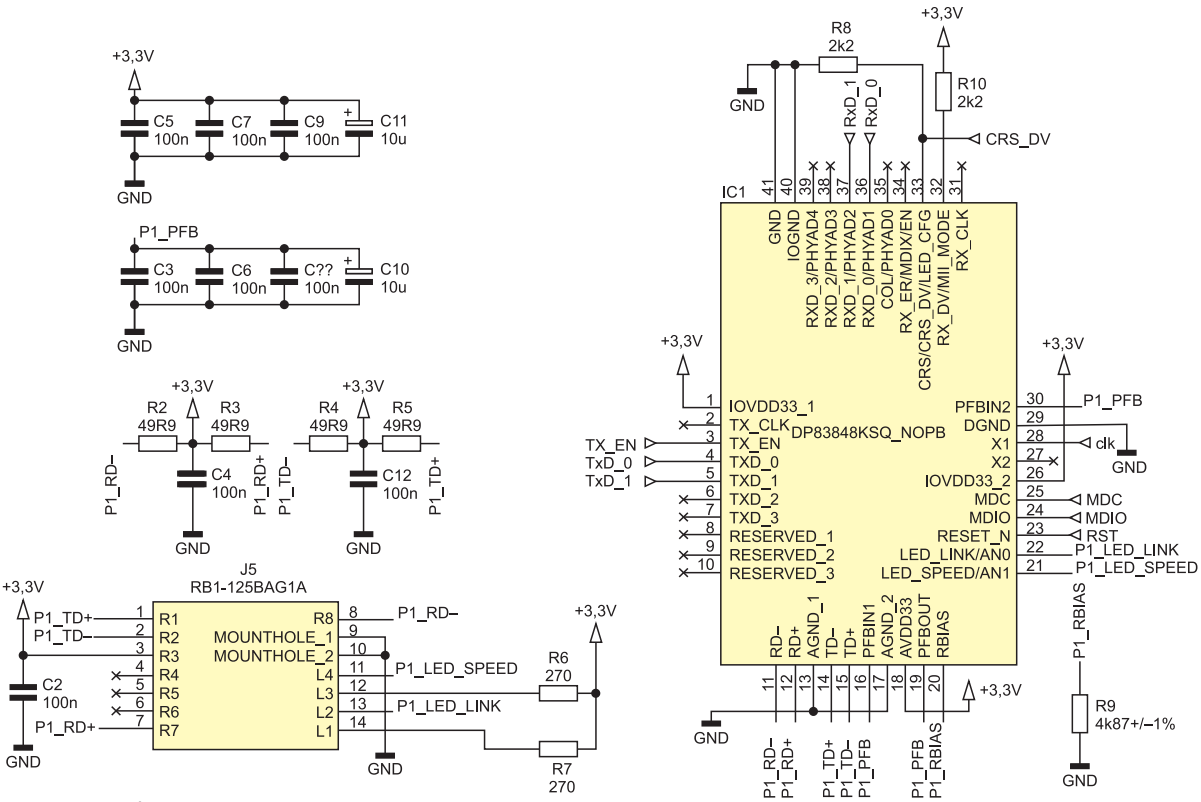
Rysunek 1. Komunikacja warstwy fizycznej z innymi warstwami

- Rx+/Rx–: różnicowa linia odbiorcza,
- Tx+/Tx–: różnicowa linia nadawcza.

Na **rysunku 2** zostały pokazuje kolejne operacje realizowane przez warstwę fizyczną. W torze nadawczym na jej wejście przekazujemy kolejne bity ramki. Pierwszym krokiem jest kodowanie 4B/5B. Polega ono na słownikowym zamienieniu kolejnych czwórek bitów na odpowiadające im piątki. Słownik jest częścią standardu Ethernet. Dzięki tej operacji w każdej z kolejnych piątek bitów będą co najmniej dwa równe 1. Następnie znajduje się skrambler, który powoduje pseudolosowe, ale odwracalne mieszanie bitów. Celem tej operacji jest zmniejszenie poziomu szumów, jaki byłby generowany przy kilkukrotnym wysyłaniu pojedynczego 4-bitowego symbolu. Kolejnym krokiem jest kodowanie NRZI, które powoduje nadanie 0, jako utrzymanie poprzedniego stanu, oraz nadanie 1, jako jego zmianę na przeciwny. Kodowanie 4B/5B gwarantuje,



Rysunek 2. Funkcja realizowane przez warstwę fizyczną



Rysunek 3. Schemat pojedynczego PHY

że w kolejnych 5-bitowych blokach wystąpią co najmniej dwie 1, a to odpowiada co najmniej dwóm zmianom znaku.

Do tej pory wykorzystywany był sygnał binarny, jednak mając linię różnicową, możemy na niej łatwo uzyskać 3 stany:

- dodatni (+) – linia dodatnia ustawiona na 1, a ujemna na 0,
- neutralny (0) – obie linie ustawione na 1 albo 0,
- ujemny (–) – linia ujemna ustawiona na 1, a dodatnia na 0.

Z tej funkcjonalności korzysta kodowanie MLT-3. Wtedy, gdy dla NRZI następuje zmiana stanu, tu również ją wykonujemy, ale przechodzimy po kolei przez stany 0, +, 0, –, 0 i tak dalej.

Odbiornik ma bardzo podobną strukturę, lecz tym razem operacje są wykonywane w przeciwnym kierunku. Znajdziemy w nim dwa dodatkowe bloki: część analogową oraz cyfrowe przetwarzanie sygnałów (DSP). Nie jest to część standardu, ale bardziej szczegół implementacyjny. Dzięki nim PHY (warstwa fizyczna) radzi sobie z zakłóceniami powstałymi w linii transmisyjnej. Poza tym warstwa fizyczna jest także odpowiedzialna za wykonanie autonegocjacji. Obie strony informują się o wspieranych rodzajach połączenia (na przykład 100 Mb/s, 10 Mb/s, full/half duplex) i negocjują najszybszą opcję obsługiwaną po obu stronach [1].

Wykaz elementów:

Rezystory:
R1, R20: 1,5 kΩ SMD0603
R2…R5, R11…R14: 49,9 Ω SMD0603
R6, R7, R15, R16: 220 Ω SMD0603
R8, R10, R17, R19: 2,2 kΩ SMD0603
R9, R18: 4,87 kΩ 1% SMD0603

Kondensatory:
C1…C9, C12…C20: 100 nF SMD0603
C10, C11, C21, C22: 10 µF (TMCP0J106MTRF)

Półprzewodniki;
IC1, IC2: DP83848KSQ/NOPB
Y1: 50 MHz DSC1004CI5-050.0000

Pozostałe:
A1: złącze Arduino (goldpin 10×1, 8×1, 6×1, 6×1)
J1, J3: goldpin 4×1
J2: goldpin 1×1
J4: goldpin 2×1
J5, J6: RB1-125BAG1A

Budowa i działanie

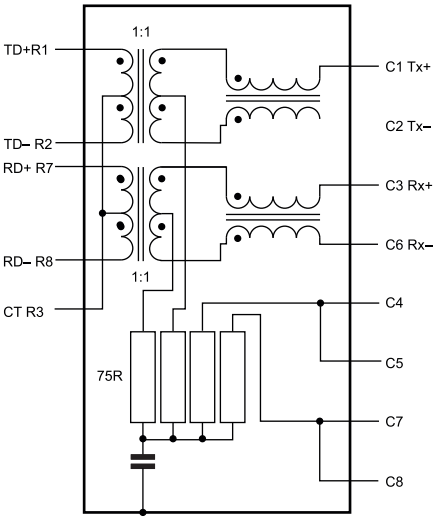
Dzięki dużej popularności, którą zdobył standard Ethernet, dostępne są układy scalone obsługujące warstwę fizyczną. Dzięki temu można ją zrealizować stosunkowo łatwo, z wykorzystaniem jedynie kilku dodatkowych elementów. Na **rysunku 3** został pokazany schemat pojedynczego toru PHY. W projekcie znajdują się dwie takie instancje.

Głównym elementem jest układ scalony DP83848 firmy Texas Instruments [2]. Po stronie MDI widzimy dwie pary rezystorów R2/R3 i R4/R5 podciągające linie transmisyjne do plusa zasilania oraz kondensatory filtrujące C4 i C12. Dalej znajdziemy złącze J5. Jest to gniazdo RJ45 z wbudowanymi transformatorami separującymi oraz sygnalizującymi diodami LED. Jego wewnętrzna budowa została pokazana na **rysunku 4**. Widzimy, że wyprowadzone są jedynie cztery linie używane przez standard Fast Ethernet. Same diody są sterowane bezpośrednio przez układ scalony. Zielona sygnalizuje wspieraną prędkość. Jest zapalona podczas pracy z prędkością 100 Mb/s, a zgaszona, gdy wykorzystana jest prędkość 10 Mb/s. Świecenie się diody żółtej informuje o wykryciu łącza. Gdy odbywa się transmisja, dodatkowo następuje jej miganie. Ten tryb pracy został włączony przez podciągnięcie za pomocą R8 pinu 33 do masy.

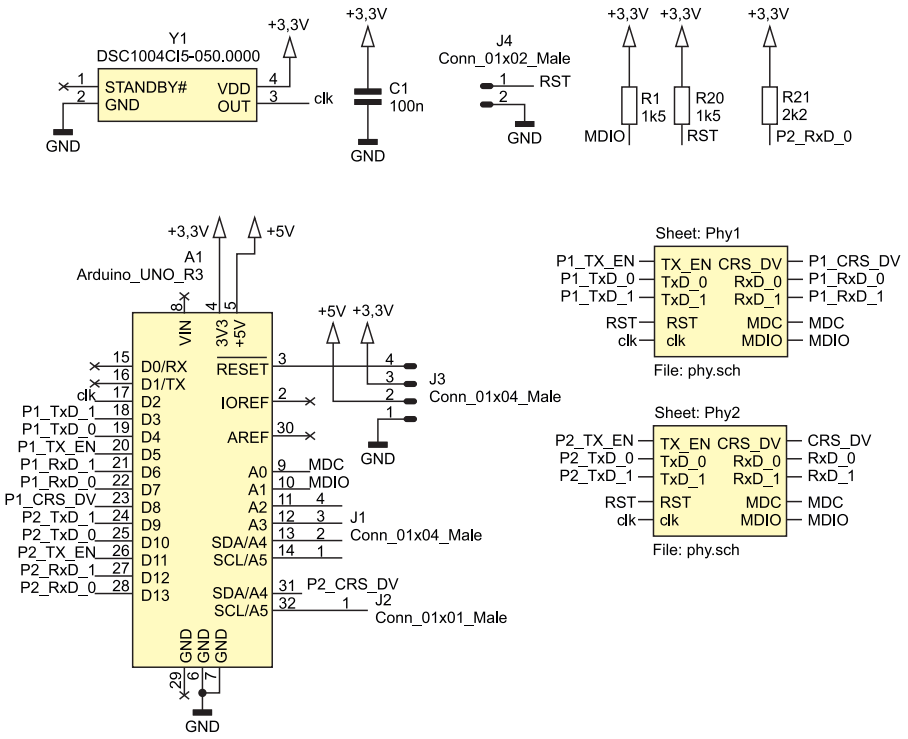
Rezystor R9 o wartości 4,87 kΩ zapewnia bias (niezerową wartość napięcia). Natomiast sygnały PFBOUT i PFBIN muszą być ze sobą połączone. Blisko nich muszą zostać umieszczone kondensatory filtrujące C3, C6, C8 i C10.

Konfiguracja układu jest wykonywana za pomocą podciągnięcia wybranych linii do masy albo plusa zasilania. Opornik R10 podciąga pin MII_MODE do plusa zasilania, dzięki czemu po starcie układ rozpocznie pracę w trybie RMII.

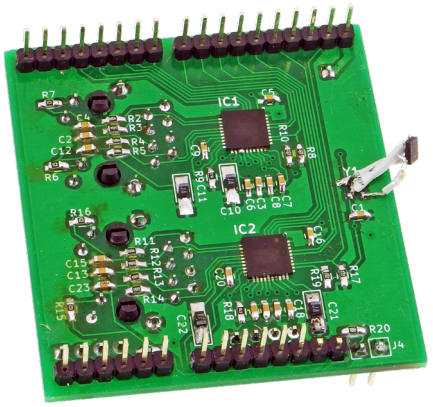
Część sygnałów została wyprowadzona do modułu nadrzędnego, są to przede wszystkim linie interfejsu RMII. Poza tym został



Rysunek 4. Wewnętrzna budowa złącza RB1-125BAG1A



Rysunek 5. Schemat ideowy modułu Rysino-net



Fotografia 1. Widok zmontowanego układu

Montaż i uruchomienie

Schemat montażowy został pokazany na **rysunku 6**. Jest to dwustronna płytka drukowana. Lutowanie najlepiej rozpocząć od elementów najmniejszych, a zakończyć na przewlekanych. Po montażu warto sprawdzić za pomocą miernika, czy nie powstały zwarcia. Zdjęcie gotowego modułu pokazuje **fotografia 1**. Generator kwarcowy jest przylutowany „na przewodach", ponieważ w pierwszej wersji płytki wkradł się błąd.

Rysunek 7 pokazuje podłączenie modułu do pinów układu FPGA w płytce Rysino. Taki schemat przyda się podczas przygotowywania projektów w środowisku Quartus. Zegar został doprowadzony do pinu 88, który jest jednym z wejść zegarowych. Interfejs podłączony do wejść D3...D8 obsługuje złącze J5, a do wejść D9...D14 złącze J6.

Po włączeniu zasilania obie zielone diody w portach RJ45 powinny się zaświecić. Po podłączeniu przewodu do aktywnego portu zaświeci się także dioda żółta. Taki sam efekt uzyskamy także poprzez połączenie obydwu portów ze sobą. Jeżeli płytka zostanie podłączona do dwuportowej karty Ethernet na komputerze pracującym pod kontrolą systemu Linux, możemy wyświetlić ich stan za pomocą rozkazu:

```
ip a
```

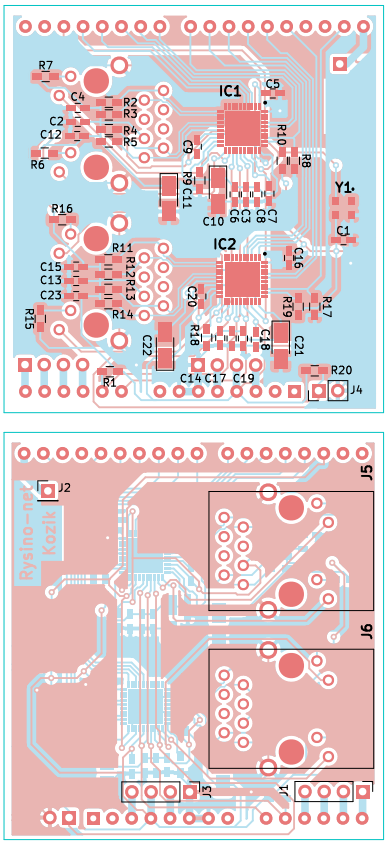
Efekt powinien być podobny do tego poniżej:

wyprowadzony interfejs zarządzania, czyli tak zwany MII Serial Management Interface (szeregowy interfejs zarządzający). Używa on dwóch sygnałów: dwukierunkowej linii danych oraz linii wyboru – *data/control*. Ostatnie dwa sygnały to zegar 50 MHz (clk) oraz reset.

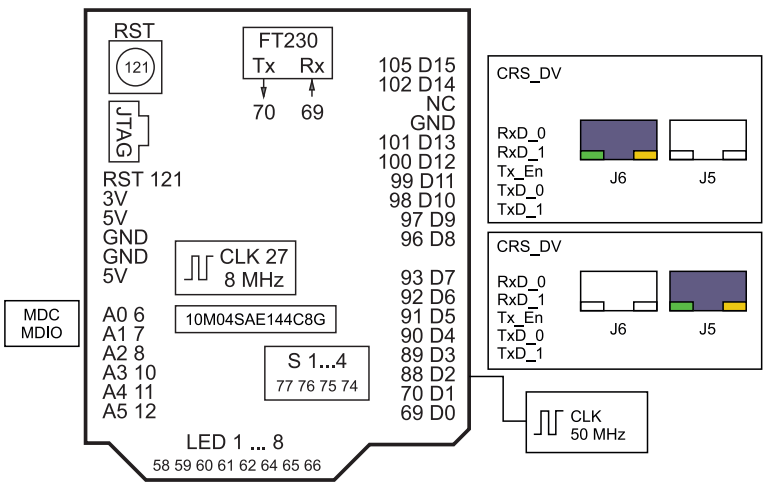
Kompletny schemat został pokazany na **rysunku 5**. Widzimy tutaj dwie instancje modułu PHY, który został pokazany na rysunku 3, oraz generator kwarcowy Y1. Sygnał zegarowy o częstotliwości 50 MHz jest doprowadzony do obu PHY oraz do magistrali RMII. Dzięki temu, że na płytce Rysino jest on podłączony do wejścia zegarowego układu FPGA, można go też użyć do jego taktowania. Takie rozwiązanie ułatwia obsługę interfejsu.

Na płytce znajduje się kilka rezystorów podciągających. R1 obsługuje linię danych interfejsu MII SMI, a R20 reset. Ważny jest także opornik R21, który poprzez podciągnięcie zerowej linii danych wejściowych w drugim PHY zmienia jej adres na interfejsie MII SMI z domyślnego 1 na 3.

Na płytce zostały wyprowadzone złącza. A1 to piny typu Arduino, poprzez które można podłączyć moduł do płytki Rysino. Na piny J1 i J2 zostały wyprowadzone wolne wejścia/wyjścia, których można użyć do własnych celów. J3 dostarcza napięć zasilania oraz sygnału reset. Zwarcie J4 spowoduje reset obu PHY.



Rysunek 6. Schemat montażowy



Rysunek 7. Podłączenie modułu do FPGA w Rysino

```
5: enp37s0f0:
<BROADCAST,MULTICAST,UP,LOWER_UP>
mtu 1500 qdisc mq state UP group
default qlen 1000
  link/ether 00:0a:f7:7e:f9:ac brd
ff:ff:ff:ff:ff:ff
  inet6
fe80::5168:ab03:68ad:519c/64 scope
link noprefixroute
    valid_lft forever preferred_
lft forever
6: enp37s0f1:
<BROADCAST,MULTICAST,UP,LOWER_UP>
mtu 1500 qdisc mq state UP group
default qlen 1000
  link/ether 00:0a:f7:7e:f9:ad brd
ff:ff:ff:ff:ff:ff
  inet6
fe80::f106:6fe4:1f53:4835/64 scope
link noprefixroute
    valid_lft forever preferred_
lft forever
```

Parametry połączenia dla wybranego portu można sprawdzić poleceniem:

```
ethtool enp37s0f0
```

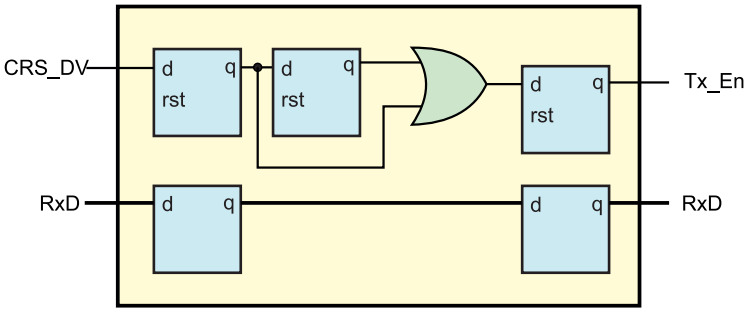
Nazwa portu może się różnić. Oto fragment uzyskanego wyniku:

```
Settings for enp37s0f0:
  Supported link modes: 10baseT/
Half 10baseT/Full
    100baseT/Half 100baseT/Full
    1000baseT/Half 1000baseT/Full
  Supports auto-negotiation: Yes
  Advertised link modes: 10baseT/
Half 10baseT/Full
    100baseT/Half 100baseT/Full
    1000baseT/Half 1000baseT/Full
  Advertised auto-negotiation: Yes
  Link partner advertised
link modes:
    10baseT/Half 10baseT/Full
    100baseT/Half 100baseT/Full
  Link partner advertised auto-
negotiation: Yes
  Speed: 100Mb/s
  Duplex: Full
  Port: Twisted Pair
  Auto-negotiation: on
  MDI-X: off
  Link detected: yes
```

Na początku widzimy prędkości i rodzaje transmisji wspierane przez kartę. Następnie w polach Link partner widzimy informacje rozgłaszane przez nasze PHY. Na końcu znajduje się opis aktywnej konfiguracji. W moim przypadku wynegocjowana prędkość transmisji to 100 Mb/s. Można ją zmienić, używając rozkazu:

```
sudo ethtool –s enp37s0f0 speed
10 duplex full autoneg off
```

Jego wykonanie spowoduje zmianę prędkości na 10 Mb/s. PHY wykryje ten fakt i zgasi zieloną diodę przy odpowiednim porcie. Przywróćmy jednak pierwotne ustawienia:



Rysunek 8. Projekt testowy rmii_echo

```
sudo ethtool -s enp37s0f0 speed
100 duplex full autoneg on
```

Projekt testowy

Do sprawdzenia działania modułu można skorzystać z projektu testowego „Echo". Znajdziemy go w repozytorium [3] w folderze board_test. Na **rysunku 8** pokazano logikę zbudowaną w tym projekcie. Ponieważ sygnał CRS_DV łączy w sobie informację o poprawności danych (*data valid*) z wykrywaniem zajętości linii (*carrier sense*), nie możemy go bezpośrednio połączyć z linią Tx_En. Zastosowana bramka LUB powoduje, że dane wyjściowe są poprawne, gdy na linii CRS_DV panuje stan wysoki w obecnym albo poprzednim kroku. Nie jest to rozwiązanie w pełni poprawne, ponieważ spowoduje, że kilka nadprogramowych danych zostanie przekazanych na wyjście. Nie powinno to jednak wpłynąć na odbieranie pakietów. Gotowy projekt składa się z dwóch takich modułów: po jednym dla każdego PHY.

W ramach testów możemy po prostu wgrać bitstream z pliku *board_test/output_files/board_test.sof*. Do przetestowania jego działania wykorzystamy bibliotekę Scapy [4]. Jest to biblioteka napisana w języku Python, która umożliwia manipulowanie pakietami. Jej najnowszą wersję można pobrać, klonując repozytorium:

```
git clone https://github.com/
secdev/scapy.git
```

Gdy kopiowanie się zakończy, wchodzimy do folderu:

```
cd scapy/
```

I uruchamiamy interaktywną konsolę:

```
sudo ./run_scapy
```

Tworzymy pakiet Ethernet zawierający przykładowy tekst i zapisujemy go w zmiennej p:

```
>>> p = Ether()/
Raw('Rysino-net echo test.
12345678901234567890123')
```

Ciąg >>> oznacza znak zachęty terminalu. Ponieważ nie wypełniliśmy pól nagłówka, zostaną one uzupełnione wartościami domyślnymi. Możemy je wyświetlić:

```
>>> p.show2()
WARNING: Mac address to reach
destination not found.
Using broadcast.
###[ Ethernet ]###
  dst= ff:ff:ff:ff:ff:ff
  src= d0:37:45:83:f2:f4
  type= LOOP
###[ Raw ]###
  load= 'Rysino-net echo test.
12345678901234567890123'
```

Sprawdźmy także jego długość:

```
>>> len(p)
60
```

Uzyskany wynik to 60 bajtów, a minimalna długość pakietu to 64. Jednak cztery brakujące bajty zostaną dodane przez kartę sieciową. Pełnią one funkcję sumy kontrolnej. Teraz możemy wysłać nasz pakiet przez wybrany interfejs. W moim przypadku jest to enp37s0f0:

```
>>> r=srp1(p, iface="enp37s0f0")
Finished sending 1 packets.
*
Received 1 packets, got 1
answers, remaining 0 packets
```

W raporcie widzimy, że pakiet został wysłany oraz że odebrano odpowiedź. W momencie wysyłania zaobserwujemy także miganie żółtej diody. Możemy wyświetlić odebrany pakiet:

```
>>> r
<Ether  dst=ff:ff:ff:ff:ff:ff
src=d0:37:45:83:f2:f4 type=LOOP
|<Raw  load='Rysino-net echo
test. 12345678901234567890123'
|>>
```

Zgodnie z oczekiwaniem jest on identyczny z tym, który wysłaliśmy. Analogiczny test można przeprowadzić dla drugiego portu. Konsolę możemy wyłączyć poleceniem:

```
>>> quit()
```

W artykule została przedstawiona budowa nakładki oraz sposób jej przetestowania. Kolejne doświadczenia zostaną opisane w cyklu „Eksperymenty z FPGA".

Rafał Kozik
rafkozik@gmail.com

Bibliografia:

[1] Spurgeon C. E., Zimmerman J., Ethernet – Biblia administratora kompendium wiedzy o sieciach Ethernet!
[2] DP83848x PHYTER Mini/LS Single Port 10/100 MB/s Ethernet Transceiver, Texas Instruments
[3] Repozytorium projektu: https://bit.ly/2FHa0Nv
[4] Scapy https://bit.ly/34cNtBu