20.Przykłady programów (c.d.)

20.1.Obsługa kontrolera *Ethernet*

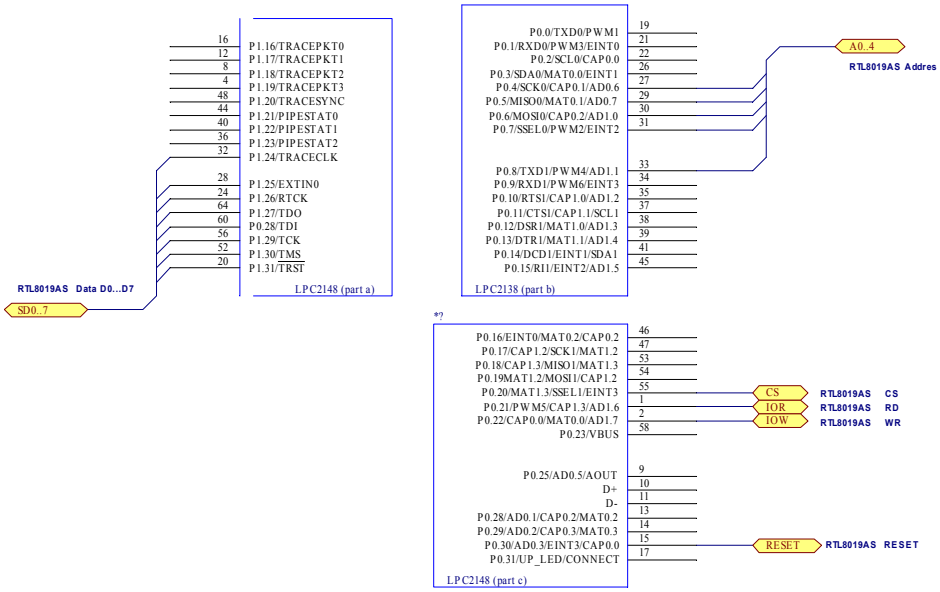
Procesory *'214x* nie posiadają wewnętrznego kontrolera sieci *Ethernet*. Wiele zastosowań mikroprocesorów wyga użycia takiego kontrolera. Ten problem można rozwiązać podłączając kontroler do portów *GPIO* procesora *'214x.*

Od wielu lat budowane są urządzenia nawet z bardzo prostymi mikroprocesorami mające dostęp do sieci *Ethernet.* Nie wszystkie istniejące rozwiązania w tej dziedzinie nadają się do zastosowania w procesorach *'214x*. Jedno z prostszych to to które stosuje firma *Microchip (www.microchip.com*) w odniesieniu do produkowanych, bardzo prostych procesorów *PIC.* Procesory te także nie posiadają zewnętrznej magistrali. Dodatkowo obsługa transmisji *Ethernet* jak i poszczególne warstwy protokołów *internetowych* napisane są bez wykorzystywania własności choćby najprostszego systemu operacyjnego oraz nie korzystają z systemu przerwań. W charakterze kontrolera *Ethernet* firma Microchip używa między innymi kontrolera *RTL8019AS*. Jest to popularny kontroler stosowany kiedyś w kartach serii *NE1000* i *NE2000.*

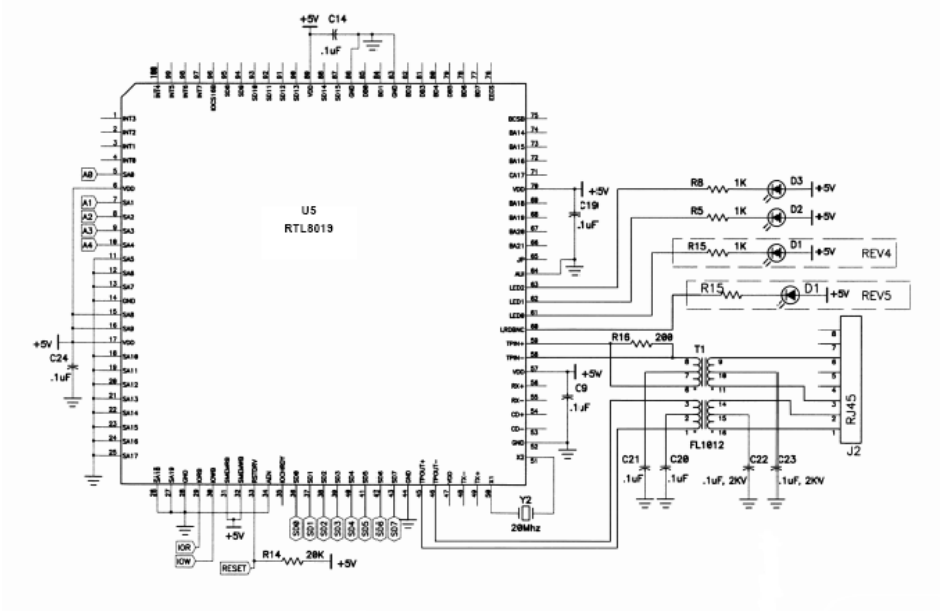
Sposób podłączenia kontrolera RTL8019AS, zwanego dalej w skrócie *RTL,* do procesora *'214x* przedstawiono na rys. 1 i 2. Do połączenia wystarczy 8-bitowa magistra danyc *SD0...7*, 5-bitowa magistrala adresowa *A0...4* oraz linie sterujące IOW (*write*) , *IOR* (*read*) , *reset* i linia *AEN* (odpowiednik *chip selec*t). Tę ostatnią linię można wyeliminować i na stałe podłączyć do sygnału masy.

Fizycznie do układu procesora można podłączyć gotowy moduł, taki jak to przedstawiono na rys. 3. Można także użyć "starej" karty *NE2000*. Na stronie *www.ethernut.de* można znaleźć opis jak przystosować taką kartę do co prawda innego procesora *AVR* ale informacja tam zawarta jest wystarczająca do zbudowania własnej konstrukcji.

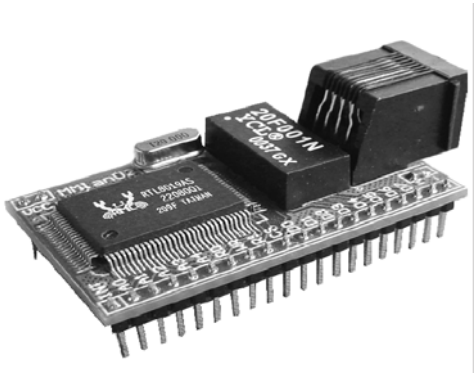
Zanim przystąpię do przedstawienia przykładowego, testowego programu jeszcze jedna uwaga na temat dokumentacji układu *RTL8019AS*. Niezbędne jest tu odwołanie do historii powstania tego układu. Układ *RTL8019AS* jest scaloną wersją wcześniejszej konstrukcji firmy *National* o nazwie *DP8930D/NS32490D*. Dlatego dokumentacja układu *RTL8019AS* nie nadaje się do studiowania i zrozumienia mechanizmów działania tego układu. Należy czytać dokumentację wcześniejszą, pochodząca od firmy *National.Semiconductors*. Układy serii *DP8930D/NS32490D* nie są już produkowane.



rys. 1 Procesor *'214x* steruje wyprowadzeniami *SD0..7, A0..4*, *IOW*, *IOR*, *AEN* i *RESET* podłączonymi do kontrolera *RTL8019AS*



rys. 2 Schemat połączeń kontrolera *RTL8019AS*. Procesor *'214x* steruje wyprowadzeniami *SD0..7, A0..4*, *IOW*, *IOR*, *AEN* i *RESET*

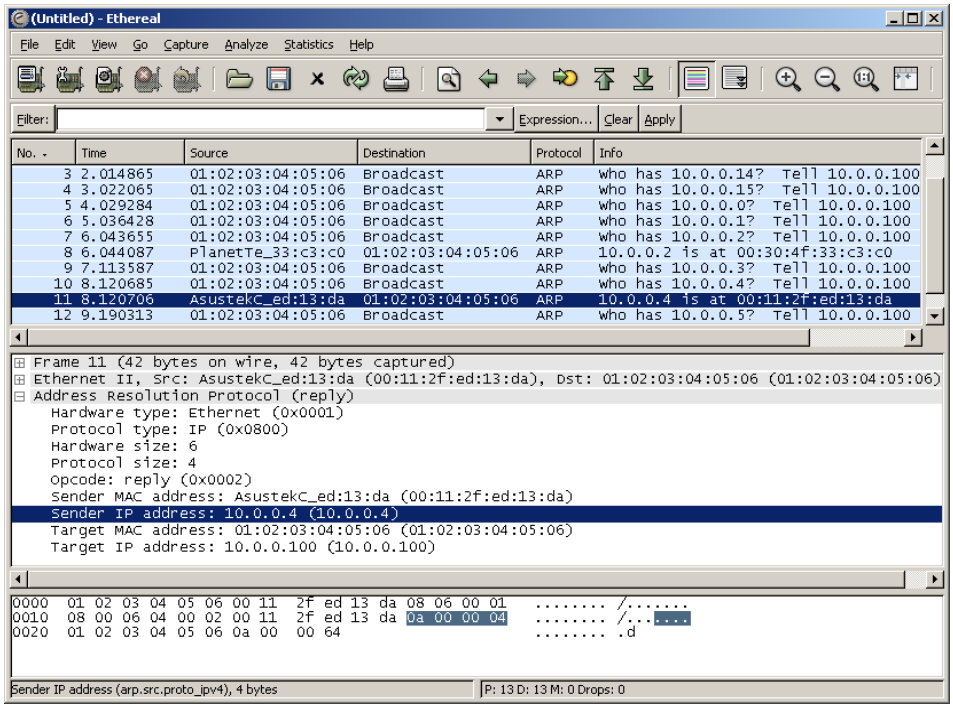


rys. 3 Moduł *RTL8019AS* firmy *Propox*

Opis przykładowego programu , pokazującego jak użyć kontroler *Ethernet* rozpocznę od pokazania rezultatów działania programu. Przestawiany przykład z konieczności musi być prosty i jednocześnie użyteczny. Zdecydowałem, że będzie to skaner ramek *Ethernet* typy *ARP* (*address resolution packet*).

Aby dokonać transmisji w Internecie gdy znany jest adres *IP* potrzeby jest jeszcze adres *MAC(media acess contoller)* karty sieciowej do której pragniemy przesłać informację. W tym celu wymyślono protokół rozgłoszeniowy (*broadcast*) *ARP*. W tym celu procesor wysyła zapytanie do wszystkich typu "hej tam uwaga, kto ma adres 10.0.0.4" (przykładowo). W odpowiedzi zwracany jest adres *MAC* karty w procesorze ze wskazanym adresem *IP*. Skanowanie będzie polegało na przeszukiwaniu kolejnych 16-tu adresów *IP*, przykładowo 10.0.0.0...10.0.0.15. Adres *MAC* odeślą tylko te procesory, które będą istniały w sieci www. Protokół *ARP* jest pierwszą fazą wykonania operacji *Ping.*

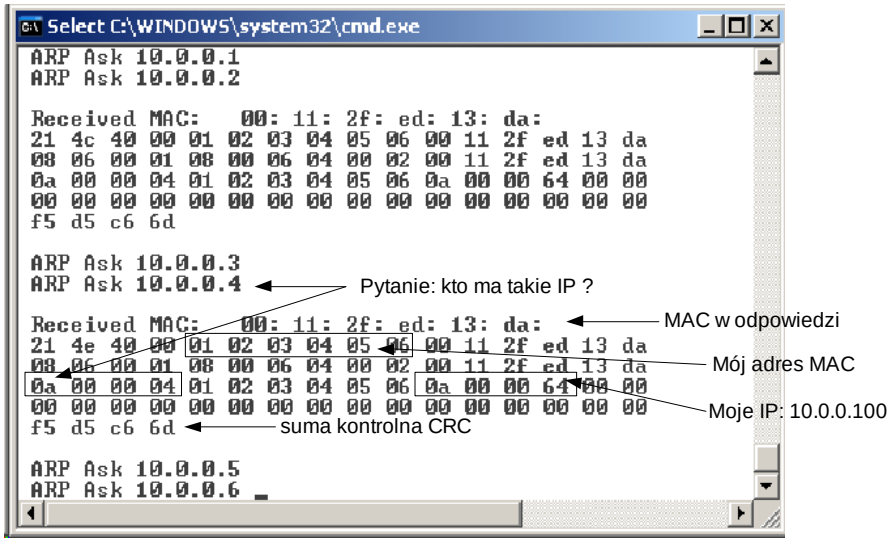
Aby można byłoobserwoć to co się dzieje w sieci Ethernet można posłużyć się analizatorem pakietów. Może nim być przykładowo program RealView. Wyniki skanowania sieci i ramki jakie pojawiają si ew sieci można zobaczyć na rys. 4.



rys. 4 Do obserwacji transmisji *Ethernet* można użyć analizatora *ETHEREAL*. Widoczna jest ramka odpowiedzi na pytanie *ARP IP*:10.0.0.4

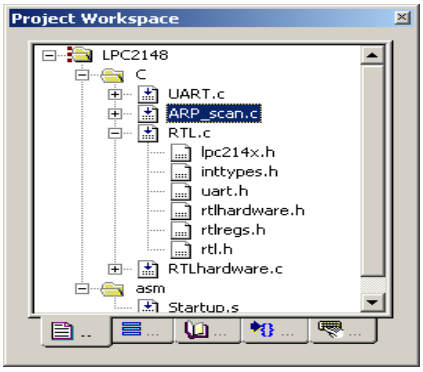
Na rysunku tym można zobaczyć kolejne zapytania o adresy *IP*. Na zapytanie kto ma adres 10.0.0.4 widoczna jest odpowiedź w dolnym okienku rysunku.

Te same wyniki produkowane są przez program *ARP_scan.c* na złączu transmisji szeregowej. Wyniki działania programu można zaobserwować w programie np. *Hyperterminial.exe*. Na rys. 5 widoczne są zapytania *ARP*. Na niektóre z nich podłączone do sieci *Ethernet* procesory odpowiadają podając zwrotnie swój adres *MAC*. Zwrotna odpowiedź widoczna na rys. 5 poprzedzona jest czterema dodatkowymi bajtami dopisanymi przez układ *RTL*. Pierwszy z nich oznacza status odebranej ramki. Liczba 0x21 oznacza poprawny odbiór. Kolejny bajt to adres bufora do którego za chwilę zostanie wpisana kolejna odebrana ramka. Na rysunku ta wartość to 0x4e. Ponieważ każdy z buforów posiada minimalną długość 256 bajtów to prawdziwy adres w pamięci to 0x4e00. Kolejne dwa bajty zwierają liczbę odebranych bajtów ramki. Wartość 0x40 to dziesiętnie 64 bajty. Jest to minimalna długość ramki w transmisji *Ethernet*. Ostatnie cztery bajty to 32-bitowa suma kontrolna *CRC*.



rys. 5 Wyniki działania programu *ARP_scan.c* . Program wysyła zapytania *ARP* a w odpowiedzi otrzymuje adresy *MAC* skanowanych urządzeń. Cztery pierwsze bajty: 21 4e 40 00 dopisywane sa przez kontroler *RTL* i oznaczają odpowiednio; status poprawnie odebranej ramki, adres następnego bufora do odbioru i długość odebranej ramki (zapis heksadecymalny)

Strukturę zbiorów składającą sie na projekt programu przedstawiono na rys.6. Przeznaczeniem projektu jest raczej wprowadzenie do protokołu *Ethernet* i zrozumienie podstaw działania kontrolera *RTL*. Mniejsze znaczenie ma w tym projekcie wartość użytkowa programu wynikowego. Ponieważ projekt jest do duży aby w pełni zrozumieć szczegóły jego działania niezbędne jest zaznajomienie się z dokumentacją kontrolera *RTL*.



rys. 6 Struktura zbiorów projektu *ARP_scan.* Program skanuje 16 adresów poczynając od zadanego adresu *IP*. Skanowany procesor w odpowiedzi podaje swój adres *MAC*

Kod głównego programu *ARP_scan.c* przedstawiono na wydruku ***kod 1***. Program najpierw inicjuje układ *RTL* (wiersz #45). Dalej wysyłana jest ramka **`NICSend`**. Odebrane ramki odpowiedzi **`NICReceive`** składowane są w buforze po to aby można

było je wydrukować. Wymienione procedury **NICSend** i **NICReceive** można łatwo zmodyfikować po to aby przesyłać inne ramki, np. *ICMP* (*ping*).

```
/*********************************************************************
ARP_scan.c                                  (skaner ARP: test Ethernet)

Program pozwala przetestować działanie kontrolera Ethernet typu
RTL8019AS. Pozwala na wysyłanie i odbiór ramek przez kontroler MAC
(Media Access Control). Program wysyła ramki rozgłoszeniowe ARP i
oczekuje na zwrotną odpowiedź zawierającą adres urządzenia MAC. Mając
adres internetowy IP poszukuje się adresu urządzenia MAC. Dla zadanego
adresu IP przeszukuje się sieć w poszukiwaniu 16-urządzeń o adresach
IP od x.x.x.0 do x.x.x.15. Odebrana ramka informacji poprzedzona jest
nagłówkiem dopisywanym przez kontroler RTL w postaci czterech bajtów.
*********************************************************************/
#include <LPC214x.H>
#include <stdio.h>
#include "inttypes.h"            // tu zdefiniowano BYTE, WORD i DWORD
#include "UART.h"
#include "RTLregs.h"                              // rejestry RTL8019AS
#include "RTL.h"           // podstawowe procedury nadawania i odbioru
#include "RTLhardware.h"   // podłączenie RTL8019AS do procesora '214x

BYTE ARP[28] = { // budowa ramki ARP --------------------------------
     00,01,                                                 // hardware
     0x08,00,                                               // ramka IP
     06,                                                 // długość MAC addr
     04,                                                  // długość IP addr
     00,01,                          // ARP request: żądanie adresu MAC
     01,02,03,04,05,06,                          // my MAC: mój adres MAC
     10,00,00,100,                                 // my IP: mój adres IP
     0xff,0xff,0xff,0xff,0xff,0xff,        // broadcast: do wszystkich
     10,0,0,4,            // target IP for resolve: szukamy MAC dla IP
};

//--------------------------------------------------------------------
void main(void) {
  BYTE x,cnt = 0;
  WORD len=68;                    // liczba odebranych bajtów z bufora
  BYTE i, frame[68];

  UART_init;                                          // przygotowanie SIO
  printf("\n Ethernet ARP scanner\n get IP=") ;
  for (i=0; i<4; i++ ) {
     scanf("%u", &x);    // podaj adres IP do skanowania, np. 10 0 0 0
     ARP[i+24]=x;
  }
  if (NICInit()) printf("\n NIC Reset Failed");                // błąd

  while (1) {
    Delay_us(1000000);
    cnt= (cnt+1) & 0x0f;
    printf("\n ARP Ask %u.%u.%u.%u",
                              ARP[24],ARP[25],ARP[26],ARP[27]+ cnt);
    NICSend(ARP[27]+ cnt);                          // wysłanie ramki ARP
    if (NICReceive(len, frame) == 0 ) {    // czekanie na odbiór ramki
        printf("\n\n Received MAC:  ");
        for (i=0; i<6; i++ ) printf(" %02x:",frame[i+10]);
        printf("\n");
        for (i=0; i<len; i++ ) {  // ramka w buforze wraz z nagłówkiem
             printf(" %02x",frame[i] ) ;
             if((i%16)==15) printf("\n");
```

```
60          }
61          printf("\n");
62       }
63     }
64  }
```

kod 1 Program *ARP_scan.c* skanuje 16 adresów *IP* poczynając ad zadanego adresu. W odpowiedzi uzyskuje się adresy *MAC* skanowanych procesorów

Procedury przedstawione na wydruku ***kod 2*** pozwalają wykonać dwie podstawowe operacje `NICGet` i `NICPut`. W procedurach tych ukryto fakt, że procesor *'214x* nie posiada magistrali adresowej i danych. Dlatego symulują one działanie magistrali poprzez sterowanie bitami portów *GPIO* procesora. Położenie bitów portów nie jest krytyczne i w doolny sposób może być zmienione. Wystarczy podać nową lokalizację dla zapisów `Nbit`.

```
1  /*********************************************************************
2  RTLhardware.c                              (procedury niskiego poziomu ...
3                                        podłączenia RTL8019AS  do procesora)
4
5  Procedury niskiego poziomu, pozwalające na obsługę układu RTL8019AS.
6  Układ RTL posiada magistralę:8 linii danych, 5 linii adresu oraz linie
7  sterujące RD, WR, CS i RESET. Takiej magistrali nie posiada procesor
8  '214x. Za pomocą procedur poniżej symuluje się działanie magistrali
9  układu RTLsterując bitami portów GPIO.
10 *********************************************************************/
11 #include <LPC214x.H>
12 #include "RTLhardware.h"
13
14 #define NBit_Data     24        // przesunięcie dla 8-miu bitów danych
15 #define NBit_Addr      4        // przesunięcie dla 5-ciu bitów adresu
16 #define NBit_reset    30                        // położenie bitu RESET
17 #define NBit_cs       20                           // położenie bitu CS
18 #define NBit_rd       21                           // położenie bitu RD
19 #define NBit_wr       22                           // położenie bitu WR
20
21 #define M           (0xFFu)            // maska dla 8-miu bitów danych
22 #define N           (0x1Fu)            // maska dla 5-ciu bitów adresu
23
24 __inline void data_to_NIC()   { IO1DIR |= M << NBit_Data ;  }   // OUT
25 __inline void data_from_NIC() { IO1DIR &= ~(M << NBit_Data);}    // IN
26
27 __inline void NIC_DATA_put(BYTE x) {                   // zapis danych
28
29   IO1CLR = ( ~x & M ) <<  NBit_Data  ;
30   IO1SET = (  x & M ) <<  NBit_Data  ;
31 }
32
33 __inline BYTE  NIC_DATA_get (void ) {                 // odczyt danych
34   return( (IO1PIN & ( M<< NBit_Data  ) ) >> NBit_Data);
35 }
36
37 __inline void addr_to_NIC()        {IO0DIR |= N << NBit_Addr;}
38 __inline void NIC_Addr_clr(BYTE x) {IO0CLR  = ( ~x & N) << NBit_Addr;}
39 __inline void NIC_Addr_set(BYTE x) {IO0SET  = (  x & N) << NBit_Addr;}
40 __inline void NIC_ADDR_put(BYTE x) {NIC_Addr_clr(x); NIC_Addr_set(x);}
41
42 __inline void setb(BYTE b)   {IO0SET  =  1 << (b);  }
43 __inline void clrb(BYTE b)   {IO0CLR  =  1 << (b);  }
44 __inline void dirb(BYTE b)   {IO0DIR |=  1 << (b);  }
45
```

```
46 #define NIC_RESET_out   dirb( NBit_reset)                  // kierunek
47 #define NIC_RESET_set   setb( NBit_reset)                     // ustaw
48 #define NIC_RESET_clr   clrb( NBit_reset)                     // zeruj
49
50 #define NIC_CS_out      dirb( NBit_cs )
51 #define NIC_CS_set      setb( NBit_cs )
52 #define NIC_CS_clr      clrb( NBit_cs )
53
54 #define NIC_RD_out      dirb( NBit_rd )
55 #define NIC_RD_set      setb( NBit_rd )
56 #define NIC_RD_clr      clrb( NBit_rd )
57
58 #define NIC_WR_out      dirb( NBit_wr )
59 #define NIC_WR_set      setb( NBit_wr )
60 #define NIC_WR_clr      clrb( NBit_wr )
61
62 #define DATA_TO_NIC     data_to_NIC()
63 #define DATA_FROM_NIC   data_from_NIC()
64
65 #define ADDR_TO_NIC     addr_to_NIC()
66
67 void NICStart(void) { //---------------------------------przygotowanie
68     DATA_FROM_NIC;
69     NIC_ADDR_put ( 0);
70     ADDR_TO_NIC ;
71
72     NIC_CS_clr;                            // układ zawsze wybrany CS=0
73     NIC_WR_set;                                 // stan początkowy WR=1
74     NIC_RD_set;                                 // stan początkowy RD=1
75     NIC_RESET_set;
76     NIC_WR_out;
77     NIC_RD_out;
78     NIC_CS_out;
79     NIC_RESET_out;
80     Delay_us(2000);
81     NIC_RESET_clr;
82     Delay_us(2000);                     // generowanie RESET RTL8019AS
83 }
84
85 void NICPut(BYTE reg, BYTE val) { //----------------------zapis do NIC
86     NIC_ADDR_put (reg);
87     NIC_DATA_put ( val);
88     DATA_TO_NIC;
89     NIC_WR_clr ;
90     Delay_us(1);                                  // krótki impuls WR=0
91     NIC_WR_set ;
92     DATA_FROM_NIC;                                   // kierunek odczyt
93 }
94
95 BYTE NICGet(BYTE reg) { //--------------------------------odczyt z NIC
96     BYTE val;
97     DATA_FROM_NIC;
98     NIC_ADDR_put (reg);
99     NIC_RD_clr;                                         // odczyt RD=0
100    val = NIC_DATA_get();                              // odczytane dane
101    NIC_RD_set;
102    return(val);
103 }
104
105 void Delay_us(DWORD us) { //-------------------------opóźnienie w usek
106    DWORD i;
107    for(i=0;i< (long)(5*us);i++){};
108 }
```

kod 2 Procedury sterujące magistralą kontrolera *RTL8019AS*. Kontroler podłączony jest do portów *GPIO* procesora *'214x*

Układ *RTL* zwiera wiele rejestrów sterujących (***kod 3***). Rejestry te pogrupowane są w strony: *page0*, *page1*... Ponieważ układ *RTL* może pracować w trybie kompatybilny z poprzednimi sterownikami *RTL* w programie wykorzystano jedynie niektóre rejestry ze strony *page0* i *page1*.

```
1  //------------------------------------------------------------------
2  RLTREGS.h                        (zbiór definicji rejestrów układu NIC)
3
4  Rejestry układu Realtek RTL8019AS. Układ ten może pracować w trybie
5  zgodnym ze standardem kontrolera NE2000.Układ RTL8019AS jest
6  zintegrowaną i unowocześnioną postacią wcześniejszego zestawu układów
7  DP8390D firmy National.
8  //------------------------------------------------------------------
9  #ifndef RTLREGS_H_
10 #define RTLREGS_H_
11
12 #define CMD        0x00                            // Command Register
13 #define IOPORT     0x10                               // I/O Data Port
14 #define RESET      0x1f                                  // Reset Port
15
16 //  Page 0 registers ------------------------------------------------
17 #define P0_PSTART 0x01              // Page Start Register for receive
18 #define P0_PSTOP  0x02               // Page Stop Register for receive
19 #define P0_BNRY   0x03                 // Boundary Pointer for receive
20 #define P0_TPSR   0x04                  // Transmit Page Start address
21 #define P0_TBCR0  0x05             // Transmit Byte Count register low
22 #define P0_TBCR1  0x06            // Transmit Byte Count register high
23 #define P0_ISR    0x07                    // Interrupt Status Register
24 #define P0_RSAR0  0x08           // Remote Start Address register low
25 #define P0_RSAR1  0x09           // Remote Start Address register high
26 #define P0_RBCR0  0x0a               // Remote Byte Count register low
27 #define P0_RBCR1  0x0b              // Remote Byte Count register high
28 #define P0_RCR    0x0c               // Receive Configuration register
29 #define P0_RSR    0x0c                      // Receive Status register
30 #define P0_TCR    0x0d              // Transmit Configuration register
31 #define P0_DCR    0x0e                  // Data Configuration register
32 #define P0_IMR    0x0f                      // Interrupt Mask register
33
34 // Page 1 registers ------------------------------------------------
35 #define P1_PAR0   0x01             // Physical Address registers 0...5
36 #define P1_CURR   0x07      // Current Page register (the next packet)
37 #define P1_MAR0   0x08             // Multicast Address register 0...7
38
39 #endif
```

kod 3 Definicje rejestrów układu *RTL8019AS*. Wymieniono tylko rejestry kompatybilne z kontrolerem *DP8390D* oraz kartą *NE2000*. Rejestry grupowane są w strony *page0...3*

```
1  /*------------------------------------------------------------------
2  RTL.h                (prototypy procedur obsługi kontrolera RTL8019AS)
3
```

```
4  Prototypy funkcji obsługi transmisji ramek Ethernet. W charakterze
5  przykładu wybrano zapytanie i odpowiedź ARP. Procedury można
6  przystosować do nadawania i odbioru dowolnych ramek Ethernet. Odbiór
7  jest ograniczony do ramek nie przekraczających 256 bajtów.
8  ----------------------------------------------------------------------*/
9  #ifndef _RTL_H_
10 #define _RTL_H_
11
12 BYTE NicReset(void);     // reset RTL powiódł się gdy odpowiedź jest 0
13 BYTE NICInit(void);                          // przygotowanie układu RTL
14 BYTE NICReceive(WORD len, BYTE *frame);           // odbiór ramki ARP
15 void NICSend(BYTE cnt);             // wysłanie ramki ARP ; cnt=0...15
16
17 #endif  //------------------------------------------------- _RTL_H_
```

kod 4 Prototypy procedur inicjowania, zapisu i odczytu ramek *Ethernet*. W charakterze prostego przykładu ramek wykorzystano ramki *ARP*. Procedury można przystosować do przesyłania innych ramek, przykładowo *ICMP*

Procedury przedstawione na wydruku ***kod 5*** pokazują jak zainicjować, wysłać i odebrać ramkę transmisji *Ethernet*. Te procedury to **`NICReset`**, **`NICInit`**, **`NICSend`** i **`NICReceive`**. Do ich zrozumienia potrzebna jest znajomość budowy kontrolera *RTL8019AS* oraz wiedza na temat przeznaczenia rejestrów. Poniżej przedstawię tylko ważniejsze fakty które ma mam nadzieję pomogą w studiowaniu zasad działania kontrolera *Ethernet.*

Budowę ramki transmisji *Ethernet* przedstawiono na rys. 7. Ramkę rozpoczyna 8-bajtów preambuły. Bajty preambuły to wartość 0xAA. Ostatni z nichch 0xAB oznacza początek transmisji *Ethernet*. Ramkę transmisji kończy 4-bajtowa wartość *CRC.* Preambuła i zakończenie CRC dodawane są automatycznie do każdej ramki przez kontroler *RTL*. Pozostałe bajty trzeba zapisać do kontrolera *RTL*. Podczas odbioru układ *RTL* odbiera tyko te ramki które adresowane są do kontrolera lub także dodatkowo ramki typu *broadcast*, tj. te które mają adres składający się z bajtów 0xFF. Odebrana ramka jest analizowana i jest poprawna jeśli zgadza się suma kontrolna *CRC*. Dalej przesyłane są adresy *MAC* urządzenia docelowego i adres *MAC* urządzenia wysyłającego. *Ethernet type* mówi o rodzaju protokołu. Wartość 0x0806 oznacza protokół *ARP*.

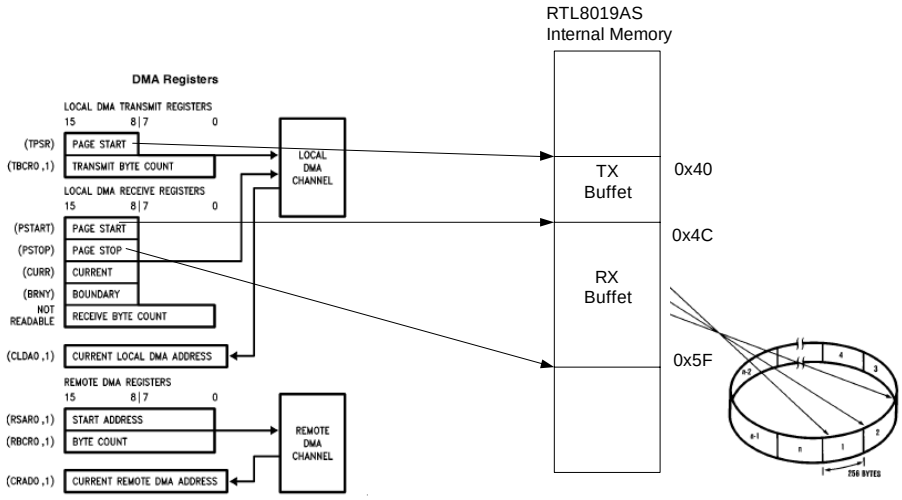
Preamble 8 bytes	0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAB
Destination MAC Address 6 bytes	0xFF, 0xFF, 0xFF, 0xFF. 0xFF, 0xFF
Source MAC Address 6 bytes	0x01, 0x02, 0x03, 0x04,0x05,0x06
Ethernet Type 2 bytes	0x08, 0x06
Data 46-1500 bytes	ARP FRAME
Frame Check sequence 4 bytes	CRC

rys. 7 Budowa ramki transmisji w standardzie *Ethernet II*. Na rysunku przedstawiono wartości jakie zostaną wysłane podczas nadawanie ramki rozgłoszeniowej *ARP* w programie *ARP_scan.c*

Zasadę działania układu RTL8019AS przestawiono na rys. 8. Układ ten zawiera we wnętrzu pamięć o pojemności 16KBajtów. W tej pamięci znajduje się bufor ramek wysyłanych *TX* oraz bufor ramek odbieranych *RX*. Układy *Local DMA* i ich rejestry wskazują ana położenie buforów w pamięci. Jednostką pamięci jest 256 bajtów.

Dlatego adres w pamięci 0x40 należy interpretować jako 0x4000. Bufor odbiorczy *RX* jest zorganizowany jako bufor cykliczny,. W buforze tym znajdują się dwa wskaźniki *CURR* i *BRNY*. Wskaźnik *CURR* pokazuje miejsce gdzie będzie zapisywana następna odebrana ramka. Wskaźnik *BRNY* pokazuje gdzie znajduje się początek aktualnie odebranych danych.

Układ *Remote DMA* jest przeznaczony do tego aby można było dokonać zapisu lub odczytu z dowolnego obszaru pamięci układu *RTL8019AS*. Odbierane lub zapisywane dane dostępne są zawsze w rejestrze *IOPORT* układu *RTL*. Zapis lub odczyt z rejestru *IOPORT* powoduje, że układ *Remote DMA* przygotuje w tym rejestrze następny bajt.



rys. 8 Podstawy działania układu *RTL8019AS*. Układ posiada 16KBajtów wewnętrznej pamięci. W pamięci tej znajduje się bufor nadawania *TX* i bufor odbioru *RX*. Bufor odbiorczy jest buforem cyklicznym ze wskaźnikami do zapisu *CURR* oraz odczytu *BRNY*. Układ Remote DMA zapisuje i odczytuje dane z pamięci i dostarcza je do procesora poprzez rejestr *IOPORT*

Wartości liczbowe pokazane na rysunku odnoszą się konkretnie po do programu *ARP_scan.c*. Odbiór ramek *Ethernet* jest bardziej skomplikowany niż nadawanie, Analizując program kodu trzeba mieć na uwadze fakt, że ramki *ARP* są krótkie i zawierają tyko 64 bajty. Jednostka pamięci układu *RTL* to 256 bajtów. Dlatego adresy które pojawić się będą w rejestrze *CURR* to 0x4C, 0x4D....0x5F, 0x4C itd. Aktualne adresy początków odebranych ramek są więc zawsze o 1 mniejsze od od zawartości rejestru *CURR*. W przypadku gdy ramki *Ethernet* są dłuższe niż 256 bajtów przedstawiona procedura **`NICReceive`** musi być zmodyfikowana.

```
1  /*********************************************************************
2  RTL.c                          (procedury obsługi transmisji Ethenet)
3
4  Przykładowe procedury obsługi kontrolera Ethernet typu RTP8019AS.
5  Procedury pozwalają zainicjować, wysłać i odebrać ramkę Ethernet. Aby
6  można było pokazać praktyczne efekty działania  wysyłane i odbierane
7  ramki to ARP (Address Resolution Packet). Ramki ARP mają prostą budową
8  i ich przesłanie poprzedza standardowe zapytanie PING. Procedura
9  odbioru ramek jest bardzo uproszczona i zakład odbiór ramek nie
10 dłuższych niż 256bajtów. Ramki ARP spełniają to kryterium bo ich
```

```
długość to 64bajty.
*********************************************************************/
#include <LPC214x.H>

#include "inttypes.h"            // tu zdefiniowano BYTE, WORD i DWORD
#include "UART.h"

#include "RTLhardware.h"                  // podstawowe procedury MAC
#include "RTLregs.h"                           // rejestry RTL8019AS
#include "RTL.h"                 // inicjowanie, zapis i odczyt ramki

#define PAGE_SIZE       0x100                 // strona to 256 bajtów
#define START_PAGE      0x40                      // początek  bufora
#define STOP_RX_PAGE    0x60                        // koniec bufora
#define TX_PAGES        6          // maksymalny rozmiar ramki 6 * 256
#define TX_BUFFERS      2                           // liczba buforów
#define FIRST_TX_PAGE   START_PAGE                 // bufor nadawania
#define FIRST_RX_PAGE   (FIRST_TX_PAGE+TX_PAGES*TX_BUFFERS)  // odbiór

extern  BYTE ARP[];                                       // ramka ARP

BYTE NICReset(void) { //------------------------------ reset RTL8019AS

    BYTE i;
    NICStart();

    for (i = 0; i < 20; i++) {
        Delay_us(500);
        NICPut(RESET,  NICGet(RESET));
        if ((NICGet(P0_ISR) & 0x80) )    return 0;   // wykonano reset
    }
    return 1;  // reset nie powiódł się
}

BYTE NICInit(void) {  //------------------------ inicjowanie RTL8019AS

    BYTE i;

    if (NICReset()) return 1;
    NICPut(CMD, 0x21);                             // MAC soft restet
    Delay_us(1600);
                                      // Data Configuration Register
    NICPut(P0_DCR, 0x48);              // 8-bitów, 2 razy 16-bit DMA
                                      // Remote Byte Count Registers
    NICPut(P0_RBCR0, 0);                                 // zerowanie
    NICPut(P0_RBCR1, 0);
                                    // Receive Configuration Register
    NICPut(P0_RCR, 0x00);           // tylko obiór ramek z adresem MAC
                                             // MAC in Loopback Mode 1
    NICPut(P0_TCR, 0x02);                       // internal loopback 1
                                    // Transmit Buffer Start Register
    NICPut(P0_TPSR, FIRST_TX_PAGE);
                                    // Initialize Receive Buffer Ring
    NICPut(P0_PSTART, FIRST_RX_PAGE);
    NICPut(P0_PSTOP, STOP_RX_PAGE);
    NICPut(P0_BNRY, STOP_RX_PAGE - 1);
                                       // Interrupt status register
    NICPut(P0_ISR, 0xff);             // kasuj wszystkie przerwania
                                          // Interrupt Mask Register
    NICPut(P0_IMR, 0x00);             // brak przerwań zewnętrznych
    NICPut(CMD, 0x61);                    // Select Page 1 and Stop
                                      // Physical Address Registers
    for (i = 0; i < 6; i++)  NICPut(P1_PAR0 + i, ARP[i+8]); // mój MAC
                                             // Multicast Registers
```

```
75      for (i = 0; i < 8; i++) NICPut(P1_MAR0 + i, 0);
76                                                  // Current Pointer
77      NICPut(P1_CURR, FIRST_RX_PAGE);
78      NICPut(CMD, 0x21);                        // Select Page 0 and Stop
79                                              // Transmit Control Register
80      NICPut(P0_TCR, 0);                    // loopback off:Set Normal Mode
81                                                  // Activate transmitter
82      NICPut(CMD, 0x22);                  // complete remote DMA: NIC start
83      return 0;
84  }
85
86  void NICSend(BYTE cnt) { //-------------------------wysłanie ramki ARP
87
88      BYTE length =60;          // minimalna ramka to 64 bajty minus CRC
89      BYTE i   ;
90                                                  // Remote Byte Count
91      NICPut(P0_RBCR0, length);             // liczba bajtów do wysłania
92      NICPut(P0_RBCR1, length >> 8);
93                                                  // Remote Start Addr
94      NICPut(P0_RSAR0, 0);
95      NICPut(P0_RSAR1, FIRST_TX_PAGE);      // adres ramki do transmisji
96
97      NICPut(CMD, 0x12);      // zapis ramki do bufora: Remote DMA Write
98
99      //  nagłowek Ethernet ============================================
100     for (i = 0; i < 6; i++)  NICPut(IOPORT, 0xFF);    // MAC Broadcast
101     for (i = 0; i < 6; i++)  NICPut(IOPORT, ARP[i+8]);       // my MAC
102
103     NICPut(IOPORT, 0x08);                         // znacznik ramki ARP
104     NICPut(IOPORT, 0x06);
105
106     //dane Ethernet ================================================
107     for (i = 0; i < 27; i++)  NICPut(IOPORT, ARP[i]);     // ARP frame
108     NICPut(IOPORT,cnt );                        // skanowanie adresów IP
109     for (i = 0; i < length-28; i++) NICPut(IOPORT, 0);  // dopełnienie
110
111     NICPut(CMD, 0x22);        // koniec zapisu do bufora: Complete DMA
112     for (i = 0; i < 20; i++)
113      if (NICGet(P0_ISR) & 0x40) break; //czekanie na koniec transmisji
114     NICPut(P0_ISR, 0x40);     // zerowanie zgłoszenia końca transmisji
115
116
117     NICPut(P0_TBCR0, (length  & 0xff));               // długość ramki
118     NICPut(P0_TBCR1, ((length >> 8) & 0xff));
119
120     NICPut(P0_TPSR, FIRST_TX_PAGE);       // adres ramki do transmisji
121     NICPut(CMD, 0x26);                            // transmituj pakiet
122 }
123
124 BYTE NICReceive(WORD len, BYTE *frame) { // odbiór ramki ARP ---------
125
126     BYTE wr,rd;
127     DWORD i;
128
129     for (i = 0; i < 500; i++)
130       if (NICGet(P0_ISR) & 0x01) goto ok;  // czekanie na odbiór ramki
131     return 1;
132
133 ok: NICPut(P0_ISR, 0x01);               // kasowanie znacznika odbioru
134     NICPut(CMD, 0x60);                            // wybór strony page1
135     wr=NICGet(P1_CURR);           // miejsce do zapisu następnej ramki
136     NICPut(CMD, 0x20);                            // wybór strony page0
137
138     NICPut(P0_BNRY,wr);
```

```
139     rd=wr-1;
140     if (rd== FIRST_RX_PAGE-1) rd=STOP_RX_PAGE - 1;
141     NICPut(P0_RSAR0, 0);
142     NICPut(P0_RSAR1, rd);         // miejsce gdzie jest ostatnia ramka
143
144     NICPut(P0_RBCR0, len & 0xFF);          // liczba bajtów do odczytu
145     NICPut(P0_RBCR1, len >>8);
146     NICPut(CMD, 0x0a);                        // odczytaj blok danych
147
148     for (i=0; i<len; i++ ) *frame++ = NICGet(IOPORT); // kolejne bajty
149
150     return 0;
151 }
```

kod 5 Zbiór *RTL.c* zawiera procedury inicjowania, zapisu i odczytu ramek *ARP*

Przedstawiony program obsługi kontrolera *Ethernet* jest wstępem do tego aby w końcu zmienić procesor *'214x* w mikroserwer internetowy, obsługujący stos protokołów *TCP/IP*. Niestety opis działania procedur obsługi stosu protokołów to przedsięwzięcie na osobną książkę. Autor wraz ze swoimi studentami przeniósł oprogramowanie *www.microchip.com* (AN833: *The Microchip TCP/IP Stack)* na procesor '214x. Sposób napisania programu obsługi *TCP/IP* i innych protokołów nadaje się do przetłumaczenia. Największy kłopot podczas tej translacji powodują struktury. Procesory firmy *Microchip* maja organizację bajtową. Podstawową jednostką procesora *'214x* jest słowo 32-bitowe. Dlatego przy dokonywaniu tłumaczenia jednostek **`struct`** potrzeba używać słowa kluczowego **`packed`** dla kompilatora *Keil ARM C.*