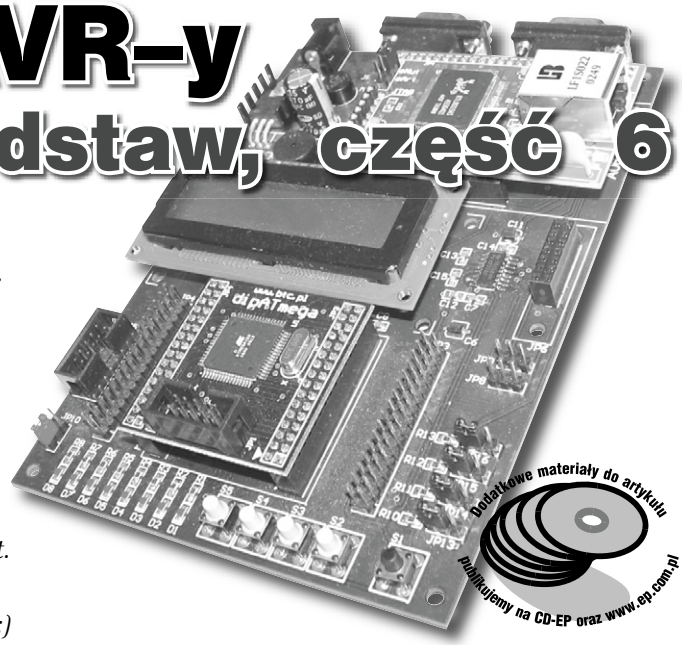
Ethernet i AVR–y
Ethernet od podstaw, część 6



W związku z niedawnymi wydarzeniami w polityce, w szóstym odcinku kursu zajmiemy się niegodziwymi lecz przydatnymi w praktyce technikami inwigilacji. Pokażę bowiem jak stworzyć program zmieniający płytkę ZL9AVR w narzędzie szpiegowskie, którego nie powstydziliby się agenci najlepszych wywiadów. Umieszczona w dyskretnym miejscu, nagrywa prowadzone w pomieszczeniu rozmowy i przesyła je odpowiednim „służbom" za pomocą sieci Ethernet. A wszystko pod przykrywką demonstracji korzystania z protokołu UDP w aplikacjach dla systemu Nut/OS :)

TCP vs UDP

Protokół UDP (*User Datagram Protocol*) jest nieco odmienny od dotychczas omawianego w kursie protokołu TCP. Zasadniczą różnicą jest bezpołączeniowość UDP. Aby przeprowadzić transmisję danych z wykorzystaniem protokołu TCP należało najpierw połączyć się z serwerem, który mógł przyjąć lub odrzucić połączenie. Dopiero potem była możliwa wymiana informacji. Protokół UDP nie posiada mechanizmów nawiązywania połączenia – transmisja danych polega na wymianie luźnych, niepowiązanych ze sobą pakietów zwanych w języku angielskim *datagramami*. O ile sesję TCP można porównać do rozmowy telefonicznej – ktoś dzwoni, druga strona odrzuca lub odbiera połączenie i następuje wymiana danych (rozmowa), o tyle transmisję danych przez UDP – do wysyłania sobie wzajemnie SMS–ów (przy czym nie wiadomo, czy druga strona przeczyta wiadomość).

Konsekwencją bezpołączeniowości UDP jest brak mechanizmów kontroli przepływu danych. Stos protokołu UDP nie potwierdza odebrania pakietu od nadawcy – nie ma zatem żadnej gwarancji, że wysłany pakiet na pewno dotrze do adresata. Jeśli pakiet zostanie zagubiony, nie jest możliwa jego automatyczna retransmisja jak w przypadku protokołu TCP (o ile nie zostanie to zaimplementowane samodzielnie).

Prostota UDP niesie ze sobą wiele korzyści: zmniejszenie opóźnienia między wysłaniem pakietu, a jego odebraniem przez adresata oraz wzrost prędkości przesyłu danych. Z tego względu jest on często używany przez gry sieciowe, oprogramowanie do wideokonferencji, telefony internetowe (VoIP) i inne aplikacje przesyłające dźwięk i obraz „na żywo".

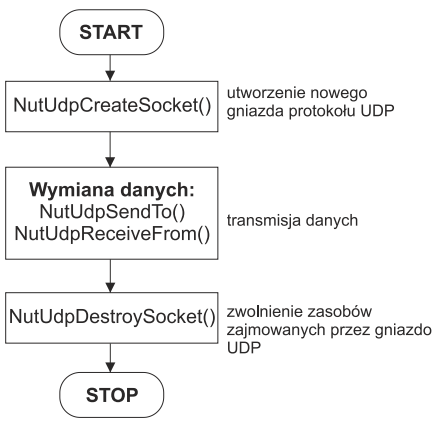
Transmisja danych przez UDP

Schemat transmisji danych z wykorzystaniem protokołu UDP w systemie Nut/OS przedstawiono na **rys. 8**. Jak zwykle zaczynamy od utworzenia nowego gniazda sieciowego, tym razem jednak będzie to gniazdo UDP – dlatego wywołujemy funkcję `NutUdpCreateSocket()`. Ponieważ protokół UDP jest bezpołączeniowy, transmisję danych możemy rozpocząć natychmiast po zainicjalizowaniu *socketa*. Aby wysłać *datagram* UDP należy skorzystać z funkcji `NutUdpSendTo()`. Funkcja `NutUdpReceiveFrom()` oczekuje na nadejście pakietu UDP, zapisuje jego treść w podanym buforze oraz zwraca adres i port hosta, który wysłał pakiet. Po zakończeniu wymiany pakietów powinniśmy zwolnić zasoby zajmowane przez gniazdo sieciowe za pomocą funkcji `NutUdpDestroySocket()`.

Prosty przykład

Jak wspomniałem na wstępie, zbudujemy dziś prosty „podsłuch" – aplikację rejestrująca sygnał z mikrofonu dołączonego do wbudowanego w mikrokontroler przetwornika A/C i wysyłającą go do komputera o wybranym przez nas adresie. Najistotniejsze fragmenty kodu źródłowego programu pokazano na **list. 10**. Główna funkcja programu ma wdzięczną nazwę – `inwigilacja()`. Rozpoczyna ona pracę od utworzenia nowego gniazda protokołu UDP. Ponieważ nie będziemy odbierać żadnych pakietów UDP, parametr `port` jest równy 0.

Następnym krokiem jest alokacja pamięci dla dwóch buforów przechowujących nagrywany dźwięk. Bufory te mają rozmiar 256 próbek, czyli 512 bajtów (próbki z przetwornika A/C są 10–bitowe, najprościej jest je zapisać w postaci liczb 16–bitowych). Alokujemy również 514–bajtowy bufor przechowujący nadawany pakiet UDP. Zastosowanie dwóch buforów umożliwia ciągłą rejestrację i transmisję sygnału – w chwili gdy dane z przetwornika A/C są wpisywane do jednego z buforów, zawartość drugiego jest wysyłana przez sieć. Gdyby wykorzystać tylko jeden bufor, podczas transmisji pakietu UDP rejestracja sygnału akustycznego nie by-



Rys. 8. Algorytm transmisji danych z wykorzystaniem protokołu UDP

łaby możliwa. Spowodowałoby to dość denerwujące „przerywanie" nagranego dźwięku.

Kolejną czynnością jest uruchomienie przetwornika analogowo–cyfrowego wbudowanego w mikrokontroler przez ustawienie odpowiednich rejestrów sterujących. Przetwornik pracuje w trybie ciągłym (*free running*) próbkując sygnał z wejścia ADC0 (pin PF0). Napięcie odniesienia jest pobierane z pinu AVCC procesora. Dzielnik zegara ADC równy 128 wymusza częstotliwość próbkowania (16 MHz/128/13 cykli zegara ADC na próbkę) =ok. 9,6 kHz wystarczającą do poprawnej rejestracji mowy.

Ostatnim etapem inicjalizacji jest ustawienie wektora przerwania ADC, wywoływanego za każdym razem gdy przetwornik zarejestruje próbkę sygnału. W systemie Nut/OS odpowiada za to funkcja `NutRegisterIrqHandler()` – nie jest zalecany tradycyjny sposób obsługi przerwań stosowany w AVR–GCC. Wektorem przerwania ADC jest w naszym przypadku funkcja `audio_record_irq()`. Odczytuje ona przetworzoną próbkę z rejestru ADCW i wpisuje ją do bufora o indeksie `active_buf`. Gdy bufor zostanie całkowicie wypełniony, zmieniamy indeks `active_buf` tak, by wskazywał na drugi bufor (`active_buf = 1-active_buf`). Dzięki temu jest możliwa ciągła rejestracja sygnału, a jeden z buforów zawsze zawiera jego kompletny fragment. Po zainicjowaniu wektora przerwania ADC możemy uruchomić próbkowanie ustawiając bit ADSC w rejestrze ADCSR.

Po inicjalizacji przechodzimy do głównej, nieskończonej pętli programu. Oczekuje ona na przetworzenie kolejnej porcji sygnału przez przetwornik A/C – gdy to nastąpi, zmieni się numer obecnie wypełnianego bufora (`active_buf`). Wówczas kopiujemy zarejestrowany fragment sygnału do bufora `tx_buffer`, który posłuży do przygotowania zawartości kolejnego *datagramu* UDP. Struktura pakietów wysyłanych przez program jest bardzo prosta – składają się one z 16–bitowego numeru pakietu (kolejne liczby całkowite), służącego do wyznaczania liczby zgubionych *datagramów* i 256–próbkowego bloku danych. Kompletny pakiet zajmuje zatem (2 bajty + 256 próbek * 2 bajty) = 514 bajtów. Po zbudowaniu pakietu, wysyłamy go do komputera tajnego agenta :) wywołując funkcję `NutUdpSendTo()`. Jego adres IP i port można ustawić za pomocą makrodefinicji `AGENT_IP` oraz `AGENT_PORT`.

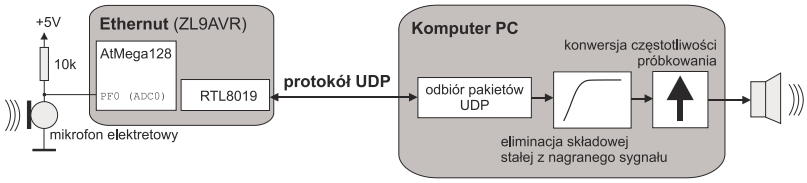
List. 10. Prosty program ilustrujący sposób korzystania z protokołu UDP

```
// adres IP komputera tajnego agenta, do ktorego bedzie transmitowana nagrana
rozmowa
#define AGENT_IP „192.168.0.2”

// port komputera tajnego agenta, do ktorego bedzie transmitowana nagrana
rozmowa
#define AGENT_PORT 12666

// liczba probek w buforz dzwieku
#define BUF_SIZE 256

// 2 bufory dla nagrywanego dzwieku. W chwili, kiedy jeden jest wypelniany
przez wektor prerwania ADC
// drugi jest wysylany przez siec
static volatile short *buffers[2];

// bufor, do ktorego obecnie zapisuje ADC oraz bufor juz wypelniony (last_ac-
tive)
static volatile int active_buf = 0, last_active_buf = 0;

// pozostala liczba probek w aktywnym buforze i indeks bufora
static volatile int samples_remaining = BUF_SIZE, buf_pos = 0;

// bufor nadawczy
unsigned short *tx_buffer;

// wektor przerwania od ADC. wpisuje kolejna probke do aktywnego bufora
static void audio_record_irq(void *arg)
{
// odczytujemy probke z rejestru przetwornika A/C
  volatile short val = inw (ADCW);

// wpisujemy ja do aktywnego bufora
  (buffers[active_buf])[buf_pos] = val;

// aktualizujemy liczniki
  buf_pos ++;
  samples_remaining --;

  if(!samples_remaining)  // koniec aktualnego bufora
  {
  active_buf = 1-active_buf;   // zmieniamy bufor na drugi
  samples_remaining = BUF_SIZE;// aktualizujemy liczniki
  buf_pos = 0;
  }

  sbi(ADCSR, ADIE);       // czyscimy flage przerwania
}


// glowna funkcja programu. Tworzy gniazdo UDP, inicjalizuje przetwornik A/C
i w nieskonczonej petli wysyla nagrane dane.
void inwigilacja()
{
  int i;

// adres IP komputera docelowego
  u_long agent_addr;

// nr obecnie przetwarzaniego pakietu
  unsigned short packet_no = 0;

// struktura gniazda UDP
  UDPSOCKET *sock;

  printf(„Rozpoczynam inwigilacje...\n”);

// tworzymy gniazdo sieciowe UDP
  sock = NutUdpCreateSocket(0);
  active_buf = 0;
  samples_remaining = BUF_SIZE;

// przetwarzamy adres odbiorcy ze stringa na liczbe 32-bitowa
  agent_addr = inet_addr(AGENT_IP);

// alokujemy pamiec na bufory do nagrywania dzwieku
  buffers[0] = NutHeapAlloc(BUF_SIZE * sizeof(short));
  buffers[1] = NutHeapAlloc(BUF_SIZE * sizeof(short));

// ... i bufor przechowujacy aktualnie wysylany pakiet
  tx_buffer = NutHeapAlloc((BUF_SIZE + 1)* sizeof(short));

// Inicjalizacja ADC

// wylaczamy konwersje ADC
  cbi(ADCSR, ADSC);

// ustawienie zrodla napiecia odniesienia z AVCC
  cbi(ADMUX, REFS1);
  sbi(ADMUX, REFS0);

// ustawienie trybu pracy (konwersja ciagla)
//  sbi(ADCSR, ADFR);

// wybor zrodla sygnalu (kanal ADC0, pin PF0)
  outb(ADMUX, inb(ADMUX) & 0xf8);
```

List. 10. c.d.

```
// ustawiamy wektor przerwania ADC.
  NutRegisterIrqHandler(&sig_ADC, audio_record_irq, NULL);

  sbi(ADCSR, ADFR);

// ustawiamy preskaler zegara ADC ( ADCCLK = MCLK / 128). Uzyskujemy w ten
sposob czestotliwosc probkowania ok. 9.6 kHz
  sbi(ADCSR, ADPS2);
  sbi(ADCSR, ADPS1);
  sbi(ADCSR, ADPS0);

// wlaczamy ADC
  sbi(ADCSR, ADEN);

// wlaczamy przerwanie ADC
  sbi(ADCSR, ADIE);

// rozpoczynamy prace przetwornika
  sbi(ADCSR, ADSC);

  for(;;) {

// jesli jeden bufor wypelnil sie, kopiujemy jego zawartosc do bufora zawie-
rajacego tarnsmitowany pakiet

  memcpy(tx_buffer+1, buffers[last_active_buf], sizeof(short) * BUF_SIZE);
  while(last_active_buf == active_buf);
// dodajemy na poczatku pakietu jego numer (uzywany przez program PeCetowy
do obliczania ilosci zgubionych pakietow)
  tx_buffer[0] = packet_no++;

// wysylamy pakiet do PC
  NutUdpSendTo(sock, agent_addr, AGENT_PORT, tx_buffer, (BUF_SIZE + 1)* si-
zeof(short));
  }

}
```

Aby cokolwiek nagrać, do przetwornika A/C należy dołączyć źródło sygnału – w najprostszym przypadku mikrofon elektretowy z rezystorem polaryzującym o wartości około 10 kΩ (patrz **rys. 9**).



Rys. 9. Tor transmisji sygnału

Oprogramowanie na PC

Do zbudowania naszego systemu podsłuchowego jest potrzebny jeszcze odbiornik – w tym przypadku prosty program dla komputera PC. Jego zadaniem jest odbiór wysyłanych przez ethernutową aplikację pakietów UDP i odtwarzanie ich treści za pomocą karty dźwiękowej. Przebieg sygnału akustycznego od mikrofonu do głośnika ilustruje rys. 9. Ponieważ mikrofon jest dołączony bezpośrednio do wejścia przetwornika analogowo–cyfrowego, konieczne jest usunięcie z sygnału składowej stałej, realizowane przez prosty filtr górnoprzepustowy.

Drugim, poważniejszym problemem jest niestandardowa wartość częstotliwości próbkowania przetwornika A/C (9,6 kHz). Aby móc poprawnie odtworzyć zarejestrowany dźwięk, program wykonuje konwersję częstotliwości próbkowania z 9,6 kHz do 22,050 kHz (standardowa częstotliwość obsługiwana przez wszystkie karty). Ze względu na małą moc obliczeniową mikrokontrolera, zadania te (filtrację i konwersję) wykonuje program na PC. Program powinien rozpocząć odtwarzanie dźwięku natychmiast po uruchomieniu Ethernuta.

Aplikacja PC–towa została napisana w języku C++ z wykorzystaniem następujących bibliotek:

- FLTK (*Fast Light Toolkit*) – interfejs użytkownika,
- PortAudio – obsługa karty dźwiękowej,
- libsamplerate – konwerter częstotliwości próbkowania,
- pthreads – obsługa wątków.

Program pracuje pod kontrolą systemów operacyjnych Linux (kompilator GCC) oraz Windows (2000/XP, kompilator Microsoft Visual C++). Do wersji windowsowej zostały dołączone odpowiednie biblioteki DLL. Źródła i binaria aplikacji dla Nut/OS oraz programu PeCetowego opublikujemy na płycie CD-EP6/2007B oraz na stronie *http://wlostowski.ep.com.pl*.

Tomasz Włostowski, EP
tomasz.wlostowski@ep.com.pl

Ważniejsze funkcje używane w przykładach

`UDPSOCKET *NutUdpCreateSocket(u_short port);`
Tworzy nowe gniazdo sieciowe protokołu UDP. Jeżeli gniazdo będzie służyć do odbierania pakietów, parametr `port` określa numer lokalnego portu, z którego dane trafiają do gniazda. Na przykład z gniazda `sock = NutUdpCreateSocket(4000)` będzie można odczytać wszystkie pakiety UDP jakie trafią na port 4000. Funkcja zwraca wskaźnik do stuktury opisującej nowo utworzonego *socketa*, albo NULL w razie niepowodzenia

`int NutUdpDestroySocket(UDPSOCKET *sock);`
Zwalnia zasoby zajmowane przez gniazdo `sock`.

`int NutUdpSendTo(UDPSOCKET *sock, u_long addr, u_short port, void *data, u_short len);`
Wysyła za pomocą gniazda `sock` pakiet UDP o zawartości znajdującej się pod adresem `data` i długości `length` bajtów do portu `port` hosta o adresie IP `addr`.

`int NutUdpReceiveFrom(UDPSOCKET *sock, u_long *addr, u_short *port, void *data, u_short size, u_long timeout);`
Oczekuje na nadejście pakietu UDP do gniazda `sock`. Po odebraniu pakietu jego dane zapisywane są pod adresem `data`. Maksymalna długość odebranego bloku danych podawana jest przez parametr `size`. Do wskaźników `addr` i `port` zapisywane są odpowiednio adres i port nadawcy pakietu. Parametr `timeout` określa po jakim czasie (w milisekundach) funkcja przerywa działanie jeśli nie odbierze żadnego pakietu. Funkcja zwraca rozmiar odebranych danych w bajtach, 0 – gdy przekroczono czas oczekiwania na odbiór pakietu lub –1 – gdy wystąpił błąd.

`int NutRegisterIrqHandler(IRQ_HANDLER irq, void (*)(void *) handler, void *arg);`
Ustawia adres funkcji `handler` (zwykła funkcja w C, bez atrybutów!) wywoływanej po wystąpieniu przerwania `irq`. Parametr `arg` jest przekazywany jako argument funkcji `handler`. Pełna listę obsługiwanych przerwań można znaleźć w pliku nagłówkowym `dev/irqreq_avr.h`.

Rys. 10. Okno aplikacji do odsłuchiwania sygnału rejestrowanego przez Ethernuta

Projekty przedstawione w cyklu artykułów o implementacjach Ethernuta uruchomiono na zestawach składających się z płyty bazowej ZL9AVR oraz modułów ZL1ETH i ZL7AVR.