Ethernet i AVR–y
Ethernut od podstaw, część 2

Miesiąc temu opisałem sposoby kompilacji i instalacji bibliotek systemu Nut/OS oraz pokazałem, jak stworzyć najprostszy program pracujący pod jego kontrolą. W drugiej części omówię funkcje pełnione przez poszczególne biblioteki Nut/OS oraz integrację Ethernuta z AVR Studio.

Biblioteki Nut/OS–a

System Ethernut składa się z kilku bibliotek i specjalnego kodu startowego, dołączanych do programu użytkownika znajdujących się (w przypadku korzystania z NutBuilda) w katalogu `winavr\NutOS\lib\avr`. Oto one:

`nutinit.o` – kod startowy systemu, uruchamiany po zerowaniu mikorkontrolera. Jego główne zadania to: inicjalizacja peryferiów mikrokontrolera (m.in. zewnętrznej pamięci RAM), ustawienie stosu, uruchomienie zarządzania wątkami. Po wykonaniu powyższych czynności, kod startowy skacze do funkcji *main()* programu użytkownika.

`libnutarch.a` – funkcje specyficzne dla używanej platformy sprzętowej (np. przełączanie kontekstu mikrokontrolera) oraz sterowniki urządzeń przeznaczone wyłącznie dla konkretnej architektury (np. UART wbudowany w mikrokontroler, kontroler Ethernet RTL8019).

`libnutdev.a` – pozostałe sterowniki urządzeń.

`libnutfs.a` – funkcje obsługi systemów plików UROM i FAT.

`libnutcrt.a` – miniaturowa wersja standardowej biblioteki języka C. Zawiera takie funkcje jak *printf()*, *malloc()*, *fopen()* itd.

`libnutos.a` – zarządzanie wątkami, obsługa komunikacji międzyprocesowej oraz funkcje dynamicznej alokacji pamięci.

`libnutnet.a` – stos TCP/IP, funkcje obsługi gniazd sieciowych (socketów).

`libnutpro.a` – obsługa popularnych protokołów sieciowych: DNS (resolvowanie nazw hostów), DHCP (automatyczna konfiguracja sieci), HTTP, FTP.

Dołączenie kodu startowego oraz bibliotek: *nutarch*, *nutos*, *nutdev* i *nutcrt* jest konieczne do prawidłowej kompilacji programów, biblioteki sieciowe (*nutnet* i *nutpro*) oraz systemu plików (*nutfs*) są opcjonalne.

Dodatkowe materiały do artykułu publikujemy na CD-EP oraz www.ep.com.pl

O makefile'ach raz jeszcze

Makefile to skrypty sterujące kompilacją programów. O tym, jak je samodzielnie tworzyć, można do-

List. 1. Kod inicjujący kontroler Ethernet i parametry TCP/IP

```
// statyczny adres IP naszego urządzenia
#define MY_IP_ADDR "10.1.108.222"
// statyczna maska podsieci
#define MY_NETMASK "255.255.0.0"
// statyczny adres domyślnej bramy
#define MY_GATEWAY "10.1.0.1"
// czy konfigurować sieć statycznie (wykomentować linię niżej), czy z użyciem
DHCP?
#define USE_DHCP

void init_network()
{
  NutRegisterDevice(&DEV_ETHER, 0, 0);

#ifdef USE_DHCP
    if(NutDhcpIfConfig("eth0", 0, 20000))
#endif
    {
        NutNetIfConfig("eth0",  NULL,
                                inet_addr(MY_IP_ADDR),
                                inet_addr(MY_NETMASK));

        NutIpRouteAdd(0, 0, inet_addr(MY_GATEWAY), &DEV_ETHER);
    }
}
```

Ethernut i AVR Studio

Aplikacje dla systemu Ethernut można również kompilować w popularnym środowisku AVR Studio. Aby utworzyć projekt wykorzystujący Nut/OS:

1. Tworzymy nowy projekt AVR–GCC.
2. Kopiujemy pliki `Makefile` i `Sources` z jednego z przykładów do katalogu, w którym założyliśmy nowy projekt.
3. Modyfikujemy plik `Sources` w następujący sposób:
   - w linii `SOURCES` umieszczamy listę plików źródłowych projektu,
   - w linii `OUTPUT` wpisujemy nazwę projektu, która musi być dokładnie taka sama jak nazwa w AVR Studio,
   - w linii `MY_CFLAGS` podajemy dodatkowe opcje dla kompilatora GCC (np. tryb optymalizacji, dodatkowe ścieżki poszukiwania plików nagłówkowych, itp.) lub zostawiamy ją pustą,
   - w linii `LIBS` podajemy biblioteki, które będą dołączone do programu (także biblioteki Nut/OS),
   - w linii `CROM_DIR` można ustawić ścieżkę do katalogu, którego zawartość znajdzie się w pamięci Flash mikrokontrolera (np. wbudowanej w urządzenie strony WWW).
4. W menu *Project–>Configuration options* zaznaczamy pole *Use external Makefile* i wybieramy skopiowany przed chwilą plik `Makefile`.

Po wykonaniu powyższych czynności, z aplikacją dla Nut/OS można pracować tak jak ze zwykłymi projektami AVR Studio. Przykładowe programy przedstawione w tym i następnych odcinkach kursu będą zawierały także gotowe projekty dla AVR Studio.

Komplet kodów źródłowych przykładów z kursu Ethernut znajdują się na płycie CD–EP1/2007B razem z najnowszą dystrybucją systemu Ethernut (4.2.1) i odpowiednimi skryptami konfiguracyjnymi.

wiedzieć się np. na stronie *http://www.eng.hawaii.edu/Tutor/Make/*. Jeśli korzystamy z plików *makefile* pochodzących z przykładowych programów z niniejszego kursu, linkowane biblioteki ustawia się w pliku `Sources` w linii LIBS. Np. aby dołączyć `libnutpro.a` należy dopisać do linii LIBS `-lnutpro`.

Istotna jest też kolejność bibliotek na liście, a w pewnych przypadkach konieczne jest powtórzenie tej samej biblioteki (wynika to ze sposobu działania linkera *ld* w przypadku, gdy biblioteki odwołują się do siebie wzajemnie), np:

```
LIBS = -lnutarch -lnutos
-lnutdev -lnutarch -lnutcrt
```

Inicjalizacja i konfiguracja interfejsu sieciowego

Zanim napiszemy jakikolwiek program wykorzystujący sieciowe możliwości systemu Nut/OS, musimy zainicjalizować kontroler Ethernet i skonfigurować parametry sieci TCP/IP. Uproszczony kod (bez obsługi błędów i listy plików nagłówkowych), który wykonuje te czynności przedstawiono na **list. 1**.

Znana z poprzedniego odcinka kursu funkcja `NutRegisterDevice` rejestruje w systemie i inicjuje kontroler Ethernet (RTL8019AS). Pojawiają się też 4 nowe funkcje, opisane w ramce.

Program pokazany na list. 1 pozwala wybrać rodzaj konfiguracji (DHCP/statycznie) za pomocą zmiany makrodefinicji `USE_DHCP`. Jeśli korzystamy z DHCP, a urządzeniu nie uda się pobrać adresu z ser-

Wyjaśnienie niektórych pojęć i terminów pojawiających się w tekście artykułu

DHCP (*Dynamic Host Configuration Protocol*) to protokół komunikacyjny umożliwiający komputerom w sieci uzyskanie od serwera danych konfiguracyjnych, np. adresu IP hosta, adresu IP bramy sieciowej, adresu serwera DNS, maski podsieci. Protokół DHCP jest zdefiniowany w RFC 2131 i jest następcą BOOTP. DHCP został opublikowany jako standard w roku 1993.

Protokół DHCP opisuje trzy techniki przydzielania adresów IP:

– przydzielanie ręczne oparte na tablicy adresów MAC oraz odpowiednich dla nich adresów IP. Jest ona tworzona przez administratora serwera DHCP. W takiej sytuacji prawo do pracy w sieci mają tylko komputery zarejestrowane wcześniej przez obsługę systemu,
– przydzielanie automatyczne, gdzie wolne adresy IP z zakresu ustalonego przez administratora są przydzielane kolejnym zgłaszającym się po nie klientom,
– przydzielanie dynamiczne, pozwalające na ponowne użycie adresów IP. Administrator sieci nadaje zakres adresów IP do rozdzielenia. Wszyscy klienci mają tak skonfigurowane interfejsy sieciowe, że po starcie systemu automatycznie pobierają swoje adresy. Każdy adres przydzielany jest na pewien czas. Taka konfiguracja powoduje, że zwykły użytkownik ma ułatwioną pracę z siecią.

Routing – jest to wyznaczenie trasy dla pakietu danych w sieci komputerowej, a następnie wysłanie go tą trasą. W sieciach opartych na protokole TCP/IP do wyznaczania trasy pakietów służy specjalna struktura zwana tablicą routingu. Zawiera ona adresy sieci IP (w postaci par adres–maska) skojarzone z prowadzącymi do nich fizycznymi interfejsami i/lub routerami. W przykładzie opisanym w artykule tablica routingu wygląda następująco:

```
Network        Gateway        Netmask        Iface
10.1.0.0       0.0.0.0        255.255.0.0    eth0
127.0.0.0      127.0.0.1      255.0.0.0      lo
0.0.0.0        10.1.0.1       0.0.0.0        eth0  ->  wpis
dodany  przez  funkcję  NutIpRouteAdd()
```

Wyznaczanie trasy dla pakietu polega na porównaniu jego adresu z kolejnymi wpisami tablicy routingu:

Jeśli `(Adres_docelowy_pakietu AND Netmask) == Network` to:

– jeśli nie jest ustawiony *gateway* –> wyślij pakiet bezpośrednio do interfejsu *Iface*,
– jeśli jest ustawiony *gateway* –> przekaż pakiet komputerowi o adresie *gateway*.

Ostatni wpis w tablicy routingu pasuje do każdego adresu pakietu i przekazuje pakiety do domyślnej bramy, czyli komputera łączącego nas z zewnętrzną siecią.

Adres 127.0.0.1 jest adresem zwrotnym urządzenia, czyli „samego siebie".

HTTP (*Hyper Text Transfer Protocol*) to protokół sieci WWW (*World Wide Web*). Właśnie za pomocą protokołu HTTP przesyła się żądania udostępnienia dokumentów WWW, informacje o kliknięciu odnośnika oraz informacje z formularzy.

Funkcje zastosowane w przykładach

`int NutDhcpIfConfig(CONST char *name, u_char * mac, u_long timeout);`

Próbuje pobrać ustawienia sieci TCP/IP z serwera DHCP i jeśli to się uda, konfiguruje z ich użyciem interfejs sieciowy oraz zwraca wartość 0. W przypadku niepowodzenia, zwracana wartość jest niezerowa.

Parametry:

name – nazwa interfejsu sieciowego, który ma być konfigurowany. W naszym przypadku – "eth0".

mac – adres sprzętowy (MAC) interfejsu sieciowego. Podanie NULL powoduje użycie domyślnego MAC–a, zapisanego w szeregowej pamięci na płytce z kontrolerem RTL8019AS.

timeout – czas w milisekundach, po upływie którego funkcja kończy działanie jeśli nie otrzyma odpowiedzi od serwera DHCP.

`int NutNetIfConfig(CONST char *name, void *mac_dev, u_long ip_addr, u_long ip_mask);`

Statyczna konfiguracja parametrów sieci TCP/IP – adresu IP i maski podsieci.

Parametry:

name – nazwa interfejsu sieciowego.

mac_dev – adres MAC interfejsu (lub NULL, jeśli ma być użyty domyślny).

ip_addr – adres IP urządzenia.

ip_mask – maska podsieci, w której pracuje urządzenie.

`int NutIpRouteAdd(u_long ip, u_long mask, u_long gate, NUTDEVICE * dev);`

Dodanie wpisu do tablicy routingu.

Parametry:

ip – adres sieci docelowej.

mask – maska sieci docelowej.

gate – brama do sieci docelowej.

dev – interfejs sieciowy, na który kierowane będą pakiety o adresach pasujących do wpisu.

`uint32_t inet_addr(CONST char *addra);`

Zamiana adresu IP w postaci ciągu znaków na liczbę 32–bitową.

Parametry:

addra – adres IP (www.xxx.yyy.zzz) w postaci łańcucha znaków.

TCPSOCKET *NutTcpCreateSocket();

Tworzy nowe gniazdo sieciowe TCP i zwraca wskaźnik do struktury reprezentującej je. W przypadku niepowodzenia, zwraca NULL.

int NutTcpAccept(TCPSOCKET *sock, u_short port);

Oczekuje na połączenie przychodzące TCP na porcie port, po odebraniu połączenia przypisuje je do gniazda siecowego sock. W razie niepowodzenia, zwracana wartość jest niezerowa.

FILE *_fdopen(int fd, CONST char *mode);

Tworzy strukturę FILE wirtualnego pliku połączonego z deskryptorem fd w trybie mode. Tryb „r+b" użyty w przykładach oznacza odczyt i uzupełnianie (r+) binarne (b). Deskryptorem fd może być np. wskaźnik do gniazda sieciowego TCPSOCKET. Zwraca wskaźnik do nowo utworzonej struktury FILE lub NULL w przypadku błędu.

void fflush(FILE *stream);

Opróżnia bufor zapisu pliku stream, powodując natychmiastowe zapisanie/wysłanie znajdujących się w nim danych.

void NutTcpCloseSocket(TCPSOCKET *sock);

Zamyka gniazdo sieciowe sock i zwalnia pamięć zajmowaną przez strukturę *TCPSOCKET*.

`void NutHttpProcessRequest(FILE *stream);`

Przetwarza połączenie HTTP przypisane do pliku–strumienia stream.

List. 2. Kod prostego serwera TCP

```
  for (;;)
    {
        char buf[256];

// tworzymy gniazdo TCP
        s = NutTcpCreateSocket();

// oczekujemy na polaczenie na port 23 (telnet)
        NutTcpAccept(s, 23);

// tworzymy plik polaczony z gniazdem s
        f = _fdopen((int) s, "r+b");

// wypisujemy tekst powitalny
        fprintf(f,"Kurs EP - Ethernut\n");
        fprintf(f,"Operacje na socketach\n");
        fprintf(f,"------------------------\n");
        fprintf(f,"Wpisz \"zgas\", aby zgasic lub \"zapal\" aby zapalic diode.\n");

// upawniamy sie, ze tekst powitalny zostanie natychmiast wyslany do klienta
        fflush(f);

// petla odbierajaca i wykonujaca polecenia tekstowe
        for(;;)
        {
// odbieramy tekst polecenia
            int len = fread(buf, 1, 256, f);

// czy koniec polaczenia?
            if(len<=0) break;

            if(!strncmp(buf,"zapal", 5))
            {
                fprintf(f,"Dioda zostala zapalona.\n");
                PORTF|=1;
            } else if(!strncmp(buf,"zgas", 4))
            {
                fprintf(f,"Dioda zostala zgaszona.\n");
                PORTF&=~1;
            };
            fflush(f);
        }

// zamykamy plik
        fclose(f);

// zamykamy gniazdo
        NutTcpCloseSocket(s);
    }
```

wera w ciągu 20 sekund, interfejs sieciowy zostanie zainicjowany parametrami statycznymi.

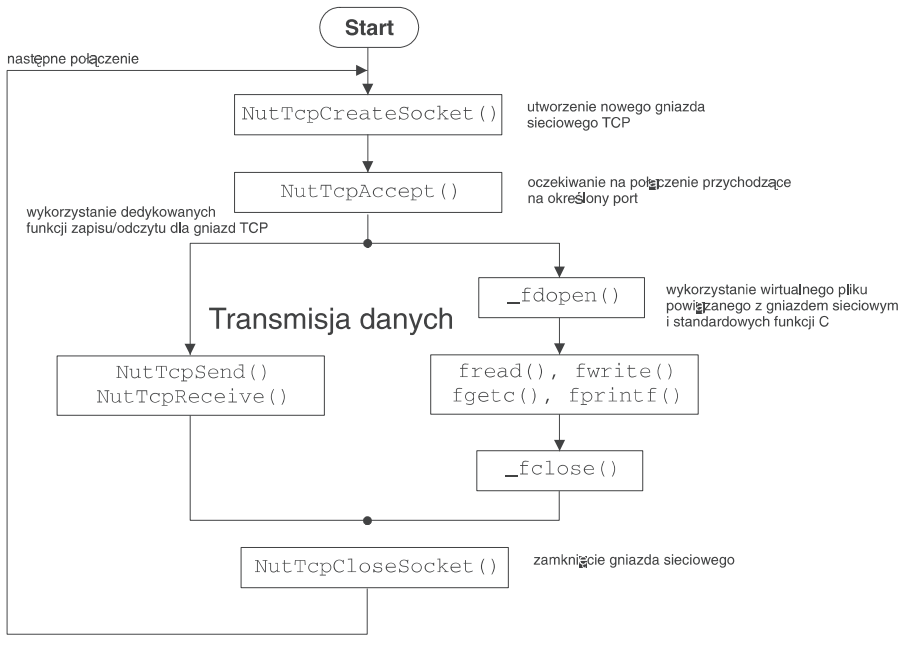
Najprostszy serwer TCP

Program, który odbiera połączenia przychodzące TCP jest nazywany serwerem, zaś taki, który inicjuje połączenia wychodzące – klientem. Nazwa „serwer" niektórym Czytelnikom może kojarzyć się ze skomplikowanym oprogramowaniem, ale w naszym przypadku implementacja prostego serwera będzie zadaniem bardzo łatwym i wartym przedstawienia na początku kursu.

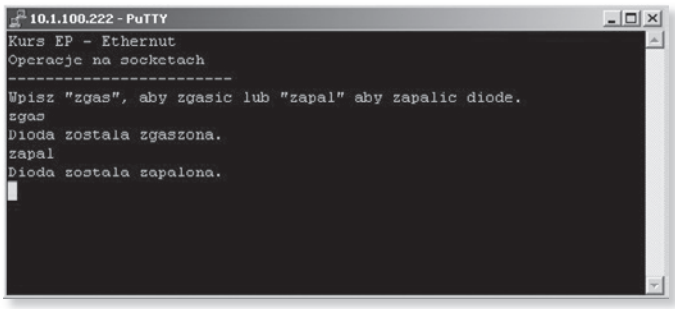
Szkielet prostego serwera TCP jest pokazany na **rys. 1**, zaś uproszczony kod w C (pozbawiony obsługi błędów) – na **list. 2.** Serwer zaczyna pracę od utworzenia nowego gniazda sieciowego (ang. *socket*) protokołu TCP (funkcja `NutTcpCreateSocket()`). Gniazdo takie jest dwukierunkowym punktem końcowym pojedynczego połączenia sieciowego. Jeśli więc mamy nawiązane połączenie TCP między dwoma urządzeniami, dane zapisane do gniazda w jednym z nich będzie można odczytać z *socketa* w drugim i na odwrót. Należy tu przypomnieć o bardzo istotnej właściwości protokołu TCP – zapewnia on integralność przesyłanych informacji, zatem dane zostaną odebrane dokładnie w takiej kolejności i postaci, w jakiej zostały wysłane (o ile istnieje fizyczne połączenie między hostami). Nie ma potrzeby stosowania sum kontrolnych, itp.

Po utworzeniu gniazda, każemy serwerowi oczekiwać na połączenie przychodzące na określony port – funkcja `NutTcpAccept()`. Porty pozawalają na odróżnienie od siebie poszczególnych serwerów działających na jednym urządzeniu, np. serwer FTP pracuje na porcie 21, serwer HTTP – 80, poczta elektroniczna – 25 (SMTP – wysyłanie) i 110 (POP3 – odbieranie). Nasz serwer będzie „słuchał" na porcie 23, przeznaczonym dla usługi Telnet. Po poprawnym odebraniu połączenia, funkcja `NutTcpAccept()` kończy działanie i zwraca wartość 0. Możemy teraz rozpocząć wymianę danych.

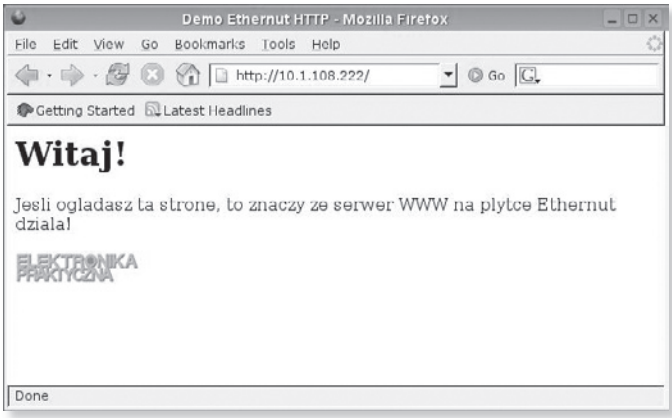
W systemie Nut/OS do wysyłania i odbierania danych z gniazda TCP służą dedykowane funkcje `NutTcpSend()` i `NutTcpReceive()`. Można jednak wykonać trick, który umożliwi traktowanie *socketa* jako zwykły plik – wówczas



Rys. 1. Szkielet serwera TCP w systemie Nut/OS



Rys. 2. Otwarta sesja *telnet* z serwerem TCP



Rys. 3. Strona testowa przykładowego serwera WWW bazującego na Nut/OS

będzie możliwe korzystanie ze znanych z języka C funkcji `fread()`, `fwrite()`, `fprintf()`, `fgetc()`, itd. Wykorzystamy w tym celu funkcję `_fdopen()`, która tworzy plik połączony z gniazdem sieciowym. Operacje we/wy na takim pliku będą powodowały zapis do/odczyt z gniazda sieciowego. Zastosowanie `_fdopen()` będzie również konieczne przy implementacji serwera WWW opisanego na końcu artykułu.

Nasz przykładowy serwer po odebraniu połączenia wysyła komunikat powitalny za pomocą funkcji `fprintf()`. Następnie w nieskończonej pętli, pobiera komendę od klienta (funkcja `fread()`), analizuje ją i zapala lub gasi diodę LED. Zwrócenie przez `fread()` wartości mniejszej lub równej 0 oznacza, że (najprawdopodobniej) połączenie zostało przerwane. Wówczas pętla wykonująca komendy zostaje zakończona, a plik–gniazdo oraz gniazdo sieciowe są zamykane (`fclose()` i `NutTcpCloseSocket()`).

W kodzie z **list. 2** pojawia się także funkcja `fflush()`. Wymusza ona wysłanie danych zgromadzonych w buforze zapisu pliku. Funkcje wykonujące operacje zapisu (np. `fwrite()`, `fprintf()`) gromadzą niewielkie ilości danych w buforach i wysyłają je „hurtem”, po napełnieniu bufora. Wywołanie `fflush()` po `fprintf()` daje pewność, że dane zapisane przez funkcję `fprintf()` zostałną natychmiast wysłane do klienta.

Została jeszcze jedna niewyjaśniona sprawa – jak połączyć się z naszym serwerem? Można do tego celu skorzystać z dowolnego klienta usługi Telnet, np. programu *Putty* (*http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html*), który wysyła do serwera teksty wpisane z klawiatury komputera, a odebrane dane odebrane wyświetla na ekranie. Otwartą sesję telnet z naszym serwerem przedstawia **rys. 2**.

Serwer WWW

System Nut/OS ma wbudowaną obsługuję protokołu HTTP, dlatego stworzenie prostego serwera stron internetowych jest bajecznie proste. Sprowadza się to do wywołania **jednej** (nie, to nie jest błąd w druku) funkcji `NutHttpProcessRequest()` po odebraniu połączenia na porcie 80 (HTTP). Kod takiego serwera (bez obsługi błędów) pokazany jest na **list. 3.**

Efekty działania serwera ethernutowego WWW przedstawiono na **rys. 3.** Oczywiście są to minimalne jego możliwości. W następnym odcinku kursu pokażę, jak obsługiwać formularze i skrypty CGI, autoryzację użytkowników oraz generowanie stron WWW z dynamiczną zawartością.

List. 3. Kod najprostszego serwera WWW

```
for (;;)
    {
        char buf[256];

// tworzymy gniazdo TCP
        s = NutTcpCreateSocket();



// oczekujemy na polaczenie na port 80 (HTTP)
        NutTcpAccept(s, 80);

// tworzymy plik polaczony z gniazdem s
        f = _fdopen((int) s, "r+b");

// przetwarzamy zapytanie HTTP
        if(f)
        {
            NutHttpProcessRequest(f);
// zamykamy plik
            fclose(f);
        }


// zamykamy gniazdo
        NutTcpCloseSocket(s);
    }
```

Pokazana tu najprostsza aplikacja serwera WWW ma jeszcze jedną wadę – może obsługiwać w danej chwili tylko jedno połączenie. Za miesiąc zapoznam Was z obsługą wątków w systemie Nut/OS, co umożliwi pokonanie tego ograniczenia.

System plików UROM

Pliki strony internetowej, którą będzie obsługiwał nasz serwer muszą być gdzieś zapisane. W przypadku bardzo prostych witryn, doskonale nadaje się do tego pamięć Flash mikrokontrolera. Twórcy Nut/OSa stworzyli w tym celu prosty system plików UROM, przeznaczony specjalnie do przechowywania niewielkich plików razem z kodem programu. Aby z niego skorzystać, należy edytować linię CRUROM_DIR w pliku `Sources`, podając ścieżkę do katalogu, którego zawartość ma znaleźć się we Flashu mikrokontrolera. W wyniku tego powstanie dodatkowy plik źródłowy `nazwa_katalogu_crurom.c` (automatycznie kompilowany i linkowany do projektu) ze strukturami systemu plików.

UROM widziany jest przez system Nut/OS jako oddzielne urządzenie, dlatego należy go zarejestrować i zainicjalizować wywołując funkcję:

```
NutRegisterDevice(&devUrom, 0, 0);
```

Tomasz Włostowski
twlostow@onet.eu

Przedstawione w artykule projekty przykładowe były uruchamiane na zestawie udostępnionym przez firmę Kamami. Dodatkowe informacje są dostępne pod adresem *www.kamami.pl*.