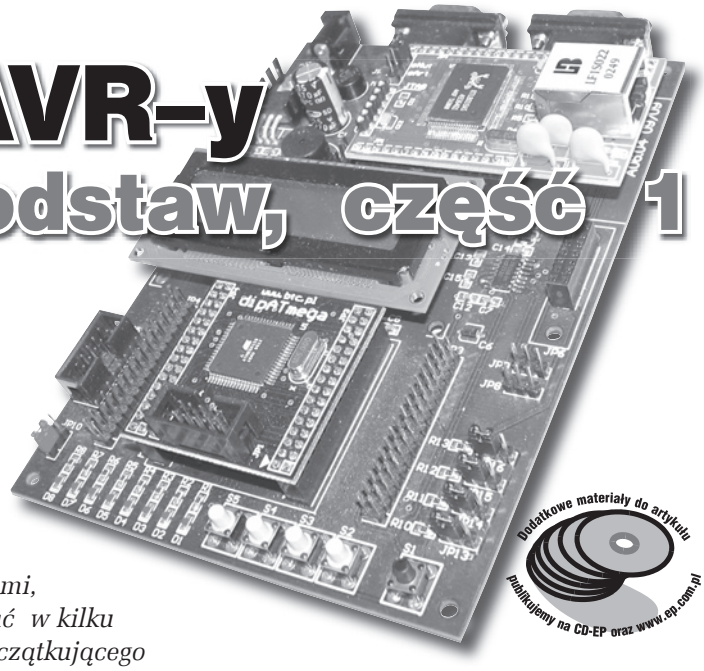
Ethernet i AVR–y
Ethernut od podstaw, część 1



Jeszcze niedawno korzystanie z sieci lokalnych i Internetu pozwalały wyłącznie duże komputery. Dziś łatwo dostępne, tanie i szybkie mikrokontrolery oraz kontrolery sieciowe umożliwiają taki dostęp również samodzielnie konstruowanym urządzeniom elektronicznym. Ethernet stał się dla nich wygodnym środkiem komunikacji i uzupełnił, a często zastąpił, dotychczas stosowane interfejsy umożliwiające wymianę danych z innymi urządzeniami, np. port szeregowy. O ile UART da się oprogramować w kilku linijkach kodu i jest to łatwe zadanie nawet dla początkującego elektronika, o tyle napisanie od początku najprostszego programu dla mikrokontrolera wykorzystującego sieć opartą na protokole TCP/IP wymaga mnóstwa czasu i wiedzy. Na szczęście nie musimy tego robić, powstały bowiem sieciowe systemy operacyjne, przeznaczone dla małych mikrokontrolerów. Jednym z nich jest Nut/OS, znany również jako Ethernut. W artykule omówię możliwości tego systemu, opiszę sposób instalacji i posługiwania się bibliotekami Nut/OS–a oraz przedstawię jego przykładowe zastosowania.

Nut/OS – ogólnie

Nut/OS to wielowątkowy system operacyjny czasu rzeczywistego (RTOS) stworzony przez niemiecką firmę Egnite. Kod źródłowy systemu jest objęty bardzo liberalną licencją BSD, co umożliwia tworzenie bazujących na nim komercyjnych aplikacji, nawet o zamkniętym kodzie źródłowym. Należy jedynie zaznaczyć, że produkt używa systemu Ethernut. Nut/OS jest dostępny dla kilku platform sprzętowych oraz obsługuje sporo urządzeń peryferyjnych, a wszystko to można łatwo dopasować do własnych wymagań. Obecnie obsługiwane są następujące rodziny mikrokontrolerów:

– Atmel AVR (Atmega103/Atmega128),
– ARM (Atmel),
– Renesas H8/300H.

System napisano w języku C z niewielką liczbą wstawek asemblerowych, bardzo przejrzyście i modularnie, dlatego stworzenie wersji na inny procesor nie powinno sprawiać kłopotów. Nut/OS udostępnia programiście typowe mechanizmy wielozadaniowego systemu operacyjnego:

– wielowątkowość z podziałem czasu procesora,
– zarządzanie pamięcią, dynamiczną alokację RAM–u,
– komunikację międzyprocesową (*events*),
– obsługę systemów plików (UROM i FAT),
– sterowniki urządzeń,
– timery (jednorazowe i okresowe),
– wysokopoziomową obsługę przerwań.

Wymienione cechy stanowią minimalną funkcjonalność jądra systemu, konieczne do uruchomienia najważniejszego elementu Nut/OSa – stosu TCP/IP Nut/Net. Oczywiście istnieją stosy, które obywają się bez wymienionych mechanizmów, ale ceną za takie uproszczenia jest niezbyt wygodny interfejs programistyczny. Dzięki integracji z systemem operacyjnym, Nut/Net jest łatwy w obsłudze, o czym przekonacie się w dalszej części artykułu.

Oto niektóre z jego możliwości:

– obsługa protokołów TCP/IP: ARP, IP, TCP, UDP, ICMP,
– obsługa kilku najpopularniejszych kontrolerów Ethernet, m.in. RTL8019AS, CS8900, LAN91C111, DM9000,
– możliwość łączenia się z Internetem za pomocą protokołu PPP (modem),
– wbudowany serwer WWW o sporych możliwościach – skrypty CGI, dynamiczne generowanie zawartości stron, autoryzacja użytkowników,
– autokonfiguracja interfejsu sieciowego z wykorzystaniem DHCP,
– wbudowane *resolvowanie* nazw *hostów*,
– bardzo prosty interfejs programistyczny, naśladujący „pecetowe” *sockety*.

Środowisko uruchomieniowe

Jako platformę do uruchamiania i testowania aplikacji Nut/OS wykorzystałem zestaw składający się z trzech modułów firmy Kamami: ZL1ETH (uniwersalny interfejs Ethernet z układem RTL8019 i gniazdkiem RJ45 zintegrowanym z transformatorem), ZL9AVR (płyta bazowa z wyposażeniem umożliwiającym budowanie własnych aplikacji ethernetowych), ZL7AVR (moduł dipAVR z mikrokontrolerem ATmega128). Kompletny zestaw pokazano na **fot. 1**.

Co to jest stos TCP/IP

TCP (*Transmission Control Protocol*) jest to strumieniowy protokół komunikacji pomiędzy dwoma komputerami. Został stworzony przez Vintona Cerfa i Roberta Kahna. Jest on częścią większej całości określanej jako stos TCP/IP. W modelu OSI TCP odpowiada warstwie transportowej. TCP zapewnia wiarygodne połączenie dla wyższych warstw komunikacyjnych zabezpieczanych za pomocą sum kontrolnych i numerów sekwencyjnych pakietów. Brakujące pakiety są obsługiwane przez żądania retransmisji. Host odbierający pakiety TCP porządkuje je według numerów sekwencyjnych tak, by przekazać wyższym warstwom modelu OSI pełen, złożony segment.
Fragment informacji z *http://pl.wikipedia.org/wiki/TCP*

Mikrokontroler ATmega128 zainstalowany na płytce ZL7AVR jest taktowany sygnałem zegarowym o częstotliwości 16 MHz. Na płytce bazowej umieszczono m.in. 32 kB statycznej pamięci RAM, niezbędnej do prawidłowej pracy systemu.

Zestaw ten jest w pełni zgodny ze specyfikacją Ethernut I, ale nieco lepiej wyposażony od wersji minimalnej: wprowadzono dodatkowe elementy często używane w systemach mikroprocesorowych – prostą klawiaturę, diody LED, złącze dla wyświetlacza alfanumerycznego LCD, 2 porty szeregowe RS232, złącze konwertera USB2RS232 oraz JTAG.

Do kompilacji projektów i bibliotek Nut/OS–a posłuży dobrze znane środowisko WinAVR oparte na AVR–owej wersji kompilatora GCC.

Kompilacja systemu

Mimo że biblioteki Nut/OS są dostępne także w formie binarnej, chciałbym na początku przedstawić sposoby kompilacji systemu ze źródeł. Zacznijmy więc od ściągnięcia kodu źródłowego ze strony *www.ethernut.de* (publikujemy je także na CD–EP12/2006B). Można pobrać zarówno wersję dla Linuksa, jak i dla Windows. Kompilację bibliotek systemu Ethernut (co wymaga wcześniejszego zainstalowania kompilatora AVR-GCC) możemy przeprowadzić na trzy sposoby, opisuję je poniżej.

Najpewniejszą metodą jest wykorzystanie skryptów dołączonych do opisanego zestawu ZL9AVR. Zawiera je archiwum *nutbuild.zip*, znajdujące się na płycie CD dołączonej do EPoL lub na stronie *kamami.pl*. Archiwum to rozpakowujemy do oddzielnego katalogu na dysku, np. `c:\nutbuild`. W tym samym katalogu umieszczamy archiwum ze źródłami systemu w wersji „for Linux Environment" – plik z rozszerzeniem *.bz2*, który zawiera sam kod źródłowy Nut/OS–a. Nasz katalog powinien wyglądać następująco:

```
[dir] data
[file] Makefile
[file] ethernut-4.2.1.tar.bz2
```

Jeżeli wersja ściągniętych źródeł Ethernuta jest inna niż 4.2.1 (najnowsza stabilna), należy edytować plik `Makefile` i zmienić zmienną `NUT_VERSION` na numer posiadanej wersji. Następnie uruchamiamy interpreter poleceń (*cmd*), wchodzimy do utworzonego przed chwilą katalogu i wykonujemy polecenia:

```
c:\nutbuild> make
```

Uruchomimy w ten sposób kompilację źródeł systemu, trwającą, w zależności od szybkości komputera, od kilkunastu sekund do kilku minut. Jeżeli wszystko przebiegnie bez zakłóceń, na ekranie zobaczymy komunikat o pomyślnej kompilacji Nut/OS-a.

```
c:\nutbuild> make install
```

Polecenie to przekopiuje skompilowane przed chwilą biblioteki, pliki nagłówkowe oraz narzędzie *crurom* do katalogu, w którym jest zainstalowany kompilator AVR–GCC (WinAVR). Biblioteki i nagłówki dla architektury AVR znajdą się w folderze `katalog_winavr\NutOS\avr`. Poprawna instalacja zostanie również zasygnalizowana odpowiednim komunikatem. Po wykonaniu powyższych czynności, katalog nutbuild możemy usunąć.

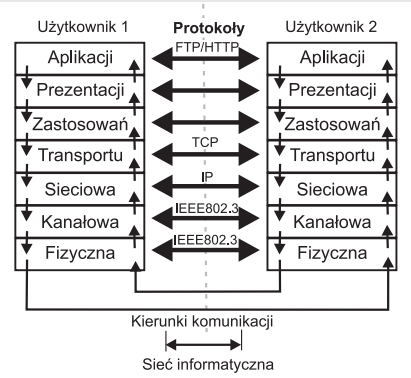
Kolejnym sposobem na zainstalowanie Nut/OS-a jest wykorzystanie graficznego konfiguratora dołączonego do źródeł systemu. W tym celu instalujemy wersję dla Windows i uruchamiamy program *Nut/OS Configurator*. Na ekranie pojawi się okno, w którym należy wybrać plik konfiguracyjny dla posiadanej przez nas platformy sprzętowej – dla zastosowanego zestawu jest to plik `ethernut13f.conf` w katalogu `katalog_instalacji\nut\conf`. Następnie w menu *Edit>Settings>Build>Platform* wybieramy interesującą nas architekturę – *avr–gcc* i z menu *Build* wybieramy pozycję *Build Nut/OS*. Jeśli wszystko przebiegnie prawidłowo, w katalogu `nutbld\lib` znajdą się biblioteki systemu. Niestety, graficzne narzędzie konfiguracyjne jest jeszcze niedopracowane, dlatego często odmawia współpracy i działa niestabilnie. Z tego powodu nie polecam jego używania.

Ostatnią opisaną metodą, najtrudniejszą, jest ręczna kompilacja systemu. W tym celu instalujemy windowsową wersję źródeł, ponieważ zawiera ona binarne narzędzia (crurom i nutconfigure). Uruchamiamy wiersz poleceń cmd i wchodzimy do katalogu, w którym zainstalowaliśmy źródła. Za pomocą programu *nutconfigure* tworzymy drzewo katalogów, w którym będziemy kompilować system dla wybranej platformy sprzętowej:

```
nutconfigure -c ./nut/conf/
ethernut13f.conf -m avr-gcc create-
buildtree
```

W wyniku wykonania powyższego polecenia powstanie katalog *nutbld*, w którym znajdą się pliki *Makefile* oraz binaria dla wybranej konfiguracji. Parametr `–c` podaje plik konfiguracyjny dla platformy sprzętowej. Plik `ethernut13f.conf` jest odpowiedni dla płyty bazowej ZL9AVR. Parametr *–m* wskazuje kompilator C (avr–gcc lub ImageCraft).

Co to jest „stos"?



Często stosowane pojęcie „stosu" określa warstwowy – jak widać na rysunku – model sieciowej komunikacji aplikacji współpracujących ze sobą zdalnie poprzez sieć informatyczną. Organizacja ISO opracowała 7–warstwowy *Model Referencyjny Połączonych Systemów Otwartych* (model OSI), który obejmuje wszystkie elementy niezbędne do zdalnej komunikacji. Przyjęto podział według sekwencji zdarzeń zachodzących podczas sesji komunikacyjnej. Warstwy od 1 do 3 zapewniają fizyczny i logiczny dostęp do sieci, a warstwy od 4 do 7 obsługują logistycznie komunikację pomiędzy aplikacjami.

Następny krok to edycja pliku `nutbld/UserConf.mk`. Dopisujemy do niego linię:

```
HWDEF+=-DNUT_CPU_FREQ=16000000
```

Jest to definicja częstotliwości oscylatora kwarcowego na płytce (16 MHz), potrzebna do obliczania wartości rejestrów sterujących UART–em i timerami. Zapisujemy plik `Userconf.mk` i w katalogu `nutbld` wykonujemy polecenia:

```
make
make install
```

W ten sposób uruchomimy kompilację systemu, która jeśli wszystko wykonaliśmy zgodnie z instrukcją, powinna przebiec bezproblemowo. Jej wynikiem będą pliki w katalogu `nutbld\lib` – biblioteki systemu Nut/OS. Należy je jeszcze zainstalować, trzeba więc wykonać kolejne czynności:

- w katalogu, w którym jest zainstalowany WinAVR (najczęściej jest to `c:\winavr`) tworzymy podkatalogi `NutOS` i `NutOS\avr`,
- kopiujemy katalog `nutbld\lib` do `NutOS\avr`,
- kopiujemy katalog `nut\include` do `NutOS\avr`,
- kopiujemy plik `nut\tools\win32\crurom.exe` do jakiegokolwiek katalogu znajdującego się w ścieżce systemowej PATH, np. `c:\winavr\bin`. Jest to narzędzie do tworzenia systemu plików

UROM, który pozwala na przechowywanie niewielkich plików w pamięci Flash mikrokontrolera (np. strony internetowej dla serwera WWW).

Dokładnie te same operacje wykonuje opisany wcześniej skrypt `Nutbuild`, wyręczając nas we „wklepywaniu" poleceń i modyfikowaniu plików konfiguracyjnych.

W taki sam sposób możemy zainstalować gotowe biblioteki zawarte w wersji dla Windows. Po skopiowaniu plików, źródła systemu (o ile nie będą nam potrzebne) możemy odinstalować. Umieszczenie bibliotek w podanym podkatalogu drzewa WinAVR jest konieczne do prawidłowego działania plików *Makefile* zastosowanych w przykładowych aplikacjach.

Mamy już zainstalowane biblioteki systemu – pora zacząć pisać programy.

Pierwszy program

Stworzymy teraz najprostszą aplikację systemu Nut/OS, będącą odpowiednikiem programu znajdującego się na początku prawie wszystkich książek o programowaniu w języku C:

```
#include <stdio.h>
main()
{
  printf("Hello, world!\n");
  return 0;
}
```

Na komputerze PC powyższy kod wyświetli oczywiście tekst: „Hello, world!". Ponieważ nasz system nie ma monitora, jako „wyświetlacz" zastosujemy terminal na komputerze połączonym z płytką przez port szeregowy. Dlatego program musi być uzupełniony o inicjalizację UART–a i przekierowanie standardowego wyjścia na port RS232. Poprawny kod jest nieco większy i wygląda następująco:

```
#include <dev/board.h>
#include <stdio.h>
#include <io.h>
#define UART_SPEED 38400
main()
{
    u_long baudrate = UART_SPEED;
    NutRegisterDevice (&DEV_DEBUG,
0, 0);
    freopen (DEV_DEBUG_NAME, "w",
stdout);
    _ioctl(_fileno(stdout), UART_
SETSPEED, &baudrate);
    printf("Hello, world!\n");
    for(;;);
}
```

Pierwsza zmiana polega na dołączeniu dodatkowych plików nagłówkowych:

– `dev/board.h`, który zawiera deklaracje nazw urządzeń oraz specyficznych dla nich struktur i typów danych,
– `io.h`, zawierający m.in. deklarację funkcji `_ioctl()`.

Jak wspomniałem wcześniej, port szeregowy wymaga inicjalizacji, nie przez ustawienie określonych wartości rejestrów sterujących UART–em, ale za pomocą interfejsu udostępnianego przez system operacyjny. Za rejestrację i inicjalizację urządzeń w systemie Nut/OS odpowiada funkcja:

```
int NutRegisterDevice(NUTDEVICE *
dev, uptr_t base, u_char irq);
```

Parametr `dev` jest wskaźnikiem do struktury opisującej sterownik urządzenia. Dla sterowników wbudowanych w Nut/OS struktury te są już zadeklarowane w plikach nagłówkowych, np. `DEV_DEBUG` odpowiada zerowemu UART–owi w mikrokontrolerze ATmega128. Przez parametry `base` i `irq` możemy przekazać adres bazowy rejestrów urządzenia i numer przerwania – w tym przypadku nie są one istotne. Jeżeli rejestracja urządzenia przebiegnie prawidłowo, funkcja `NutRegisterDevice` zwróci wartość 0, jeśli nie, wartość –1.

Następnym krokiem jest przekierowanie standardowego wyjścia (`stdout`) do portu szeregowego. Odpowiada za to funkcja:

```
FILE *freopen(CONST char *name,
CONST char *mode, FILE * stream);
```

Otwiera ona plik o nazwie `name` w trybie `mode` i (w przeciwieństwie do popularnej funkcji `fopen`) ustalonym deskryptorze stream. W naszym przypadku jest to plik o nazwie `DEV_DEBUG_NAME (uart0)` w trybie do zapisu `(w)` i deskryptorze `stdout` (standardowe wyjście). Podobnie jak w systemach uniksowych, w Nut/OSie urządzenia są reprezentowane przez pliki, dlatego zapis do pliku o nazwie `uart0` spowoduje wysłanie danych do zerowego portu szeregowego. Jeżeli funkcja `freopen` wykona się prawidłowo, zwróci wskaźnik do podanego w parametrach deskryptora pliku, w przeciwnym przypadku – pusty wskaźnik `(NULL)`.

Kolejna linia kodu wywołuje funkcję:

```
int _ioctl(int fd, int cmd, void
*buffer);
```

Służy ona do wykonywania niestandardowych operacji na urządzeniach wejścia–wyjścia. W naszym przypadku `_ioctl` odpowiada za ustawienie prędkości przesyłania danych przez port szeregowy. Parametr `fd` to numer deskryptora pliku urządzenia (w naszym przypadku uzyskaliśmy go za pomocą makra `fileno` ze struktury `FILE* stdout`), `cmd` – kod polecenia, czyli `UART_SETSPEED`, `buffer` – dane dla polecenia, w naszym programie wskaźnik do zmiennej z podaną prędkością UARTa. Poprawne wykonanie funkcji jest sygnalizowane zwróceniem wartości 0, niepoprawne, ujemnej.

Po wywołaniu powyższych funkcji, możemy wreszcie wypisać tekst: „Hello, world!". Następnie każemy programowi czekać w nieskończonej pętli.

Kompilacja programu

Kompilacja aplikacji Nut/OS wymaga pewnych dodatkowych opcji dla kompilatora C oraz podania w odpowiedniej kolejności bibliotek systemu oraz specjalnego kodu startowego (`nutinit.o`) przy linkowaniu. Chcąc ułatwić pracę czytelnikom, przygotowałem gotowe pliki sterujące kompilacją – `Makefile` i `Sources`. Plik `Makefile` zawiera reguły sterujące wywoływaniem AVR–GCC oraz opcje konieczne do prawidłowej kompilacji programu i zazwyczaj nie trzeba go modyfikować. Zmian wymaga natomiast plik `Sources`, który dla programu podanego przed chwilą wygląda następująco:

```
OUTPUT = hello
SOURCES = hello.c
OPTIONS = -Os
LIBS = -lnutarch -lnutdev -lnutos
-lnutarch -lnutcrt
```

Definicja `OUTPUT` to nazwa pliku wynikowego (bez rozszerzenia), `SOURCES` – wykaz plików źródłowych rozdzielony spacjami, `OPTIONS` – dodatkowe opcje kompilatora, w tym przypadku – optymalizacja kodu pod względem wielkości `(-Os)`. `LIBS` jest listą bibliotek dołączonych do projektu. Projekt kompilujemy wydając polecenie `make`. W jego wyniku powstaną pliki `hello.hex` i `hello.bin`, którymi możemy zaprogramować mikrokontroler.

Polecenie `make clean` usuwa wszystkie pliki utworzone podczas kompilacji, zaś `make isp` programuje procesor z wykorzystaniem programatora STK–200 (np. ZL2PRG) i programu AVRdude standardowo obecnego w środowisku WinAVR.

Za miesiąc opiszę dokładniej pliki `Makefile`, funkcje poszczególnych bibliotek systemu Nut/OS oraz przedstawię prostą aplikację serwera WWW. Zapraszam do lektury!

Tomasz Włostowski
twlostow@onet.eu