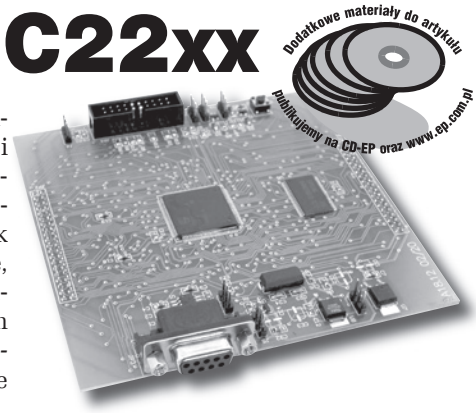
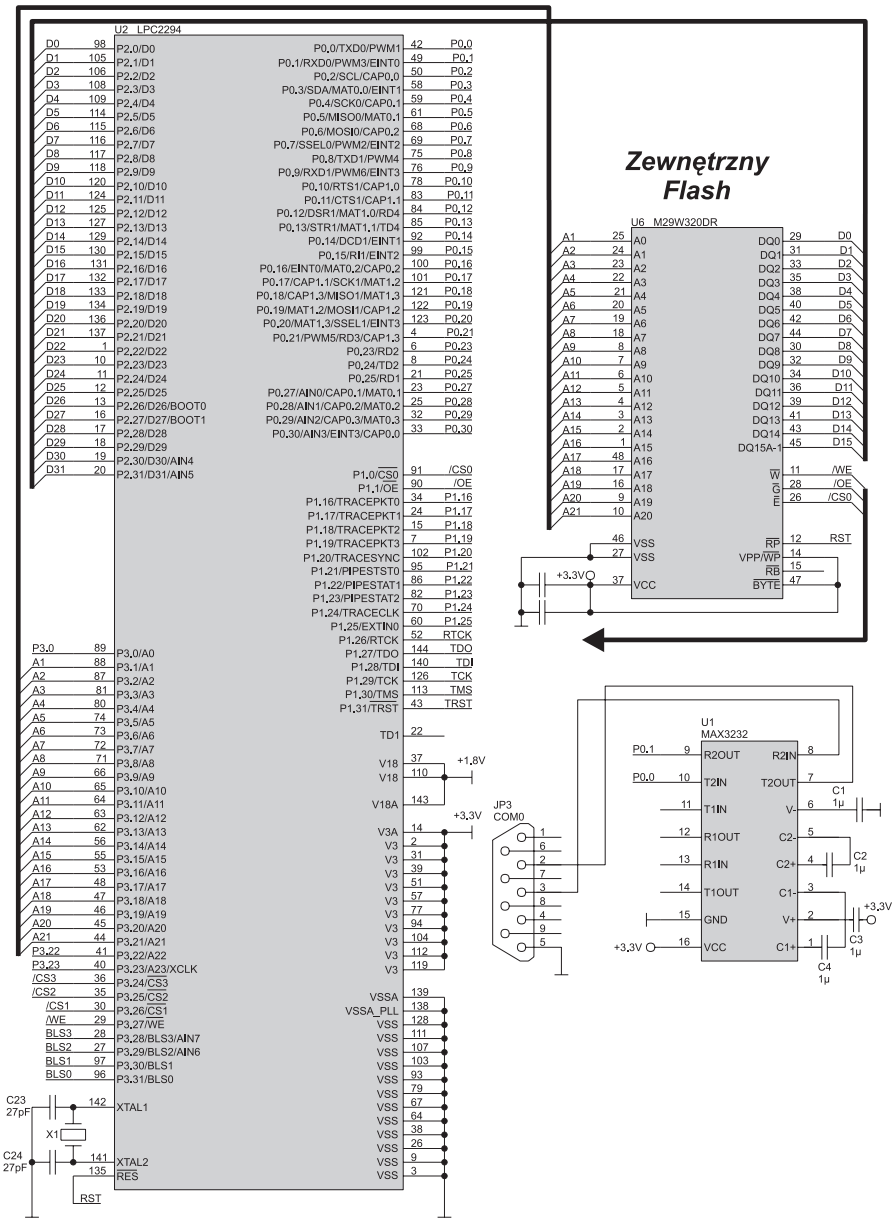
Botloader dla zewnętrznych pamięci Flash mikrokontrolerow LPC22xx

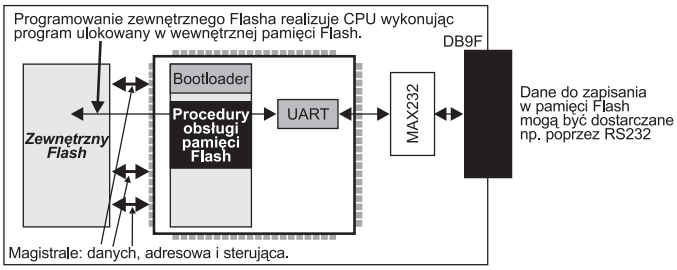


Technika programowania mikrokontrolerów z wykorzystaniem specjalnego (rezydentnego) programu – bootloadera – jest dobrze znana i ceniona wśród praktyków, gdyż nie wymaga stosowania specjalnych programatorów. Opisywany poniżej bootloader ma dodatkowo tę zaletę, że umożliwia ładowanie programu do zewnętrznej pamięci Flash dostępnej dla mikrokontrolera z rodziny LPC22xx.

Mikrokontrolery rodziny LPC2xxx zawierają wewnątrz pamięć Flash o pojemności od 32 do 512 kB oraz pamięć RAM o wielkości 8...32 kB. Zawierają również pamięć stałą ROM, w której jest umieszczony program *bootladera* umożliwiający zaprogramowanie wewnętrznej pamięci Flash w systemie, poprzez port szeregowy. Producent mikrokontrolerów dostarcza również aplikację dla systemu Windows, za pomocą której w prosty sposób możemy załadować plik z własnym programem do pamięci Flash. Dzięki temu posunięciu mikrokontrolery rodziny LPC odniosły duży sukces rynkowy, ponieważ niepotrzebne są żadne specjalistyczne przyrządy służące do programowania. Przypomnijmy, że np. w przypadku mikrokontrolerów STR711 pamięć Flash może być zaprogramowana jedynie za pomocą JTAG–a. Dla większości prostych i średnio zaawansowanych aplikacji wielkość wewnętrznej pamięci Flash i RAM jest wystarczająca, jeżeli jednak chcemy uruchomić jakąś aplikację pod kontrolą systemu operacyjnego, okazuje się, że wewnętrzna pamięć Flash i RAM jest stanowczo zbyt mała. Musimy wówczas sięgnąć, po któryś z mikrokontrolerów rodziny LPC22xx i wyposażyć go w zewnętrzną pamięć RAM i Flash. Taki projekt był już prezentowany w EP3 i 4/2006 pt. „ARMputer" (**rys. 1**). Niestety *loader* umieszczony we wnętrzu mikrokontrolera LPC22xx umożliwia zaprogramowanie tylko wewnętrznej pamięci Flash. Istnieje możliwość zaprogramowania pamięci zewnętrznej za pomocą odrębnego programatora, jednak nie jest to rozwiązanie zbyt wygodne, zwłaszcza na etapie pisania oprogramowania. Dużo lepszym rozwiązaniem jest wykorzystanie dodatkowego *loadera*, który umożliwia zaprogramowanie



Rys. 1. Schemat elektryczny ilustrujący sposób dołączenia zewnętrznej pamięci Flash do mikrokontrolera LPC22xx



Rys. 2. Ilustracja zasady działania *bootloadera* programującego zewnętrzną pamięć Flash dołączoną do mikrokontrolera LPC22xx

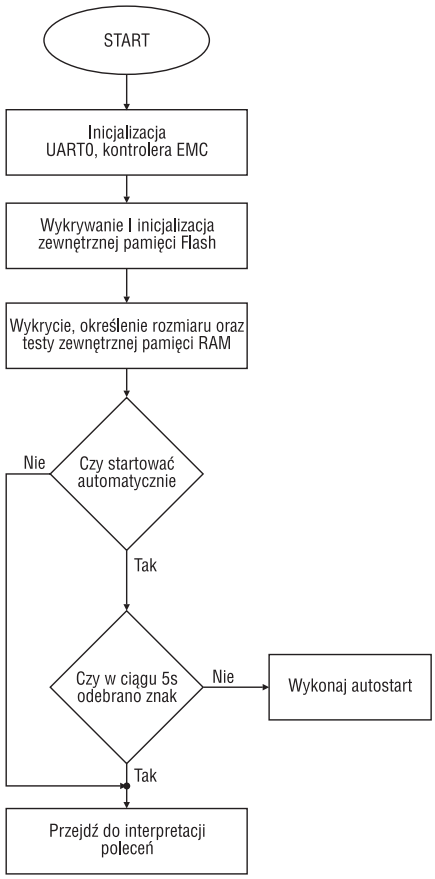
zewnętrznej pamięci Flash oraz uruchamianie kodu programu z zewnętrznej pamięci. Tematem niniejszego artykułu będzie właśnie uniwersalny bootloader/monitor (LBMON) dla mikrokontrolerów LPC22xx, który można wykorzystywać do programowania oraz uruchamiania programów pracujących w zewnętrznej pamięci mikrokontrolera. Umożliwia on także przeprowadzenie podstawowej konfiguracji układów peryferyjnych przed startem właściwego systemu operacyjnego lub programu użytkownika. Jeżeli mamy wystarczającą ilość RAM–u i Flasha, za jego pomocą możemy uruchomić chociażby system uCLinux.

Zasada działania i funkcje *loadera* LBMON

Bootloader jest małym programem umieszczonym w pamięci mikrokontrolera. Jego zadaniem jest inicjalizacja podstawowych urządzeń peryferyjnych oraz załadowanie systemu operacyjnego lub programu użytkownika z odpowiedniego nośnika, na przykład dysku, pamięci Flash, itp. W przypadku, gdy jest to nośnik niewymienialny (pamięć Flash), powinien on umożliwiać zmianę kodu programu zawartego w tej pamięci. W LBMON *loaderze* jako pamięć dla programu uruchamianego będziemy wykorzystywać zewnętrzną, równoległą pamięć NOR Flash (M29W320 **rys. 2**), natomiast pamięć wewnętrzna będzie wykorzystana tylko na potrzeby *bootloadera*. Umieszczenie *bootloadera* w pamięci wewnętrznej, zapewnia jego prawidłowe działanie, niezależnie od tego czy pamięć zewnętrzna funkcjonuje prawidłowo oraz chroni przed jego przypadkowym skasowaniem lub zamazaniem. Komunikacja z *boootloaderem* odbywa się poprzez port szeregowy, za pomocą komend tekstowych z ramką 115200,n,8,1, tak więc nie jest potrzebne specjalistyczne oprogramowanie po stronie komputera PC. Wystarczy zwykły terminal, na przykład „Hyperterminal" lub „minicom". Algorytm działania *bootloadera* przestawiono na **rys. 3**.

Po włączeniu napięcia zasilającego b*ootloader* inicjalizuje układy peryferyjne niezbędne do działania mikrokontrolera takie jak: port szeregowy, kontroler pamięci zewnętrznej EMC (*External Memory Controller*). Wykrywa rozmiar zewnętrznej pamięci RAM i Flash oraz wykonuje test pamięci RAM. W *bootloaderze* przyjęto założenie, że pamięć Flash ma organizację 16–bitową i będzie się znajdować zawsze w obszarze 0x80000000 (Bank0). Pamięć RAM (o organizacji 32–bitowej) jest natomiast umieszczona pod adresem 0x81000000 (Bank1), co jest prawidłowe dla większości urządzeń z mikrokontrolerem LPC22xx. Konfiguracja przestrzeni adresowej może być zmieniona na etapie kompilacji *bootloadera*. Po wykonaniu testów pamięci LBMON sprawdza ostatni sektor zewnętrznej pamięci Flash, w którym mogą znajdować się polecenia do automatycznego wykonania podczas uruchamiania. Jeżeli są one niedostępne, wówczas program przechodzi do interpretacji komend z terminala, natomiast jeżeli komendy startowe zostały wpisane, wówczas są one po kolei wykonywane. *Bootloader* przed automatycznym wykonywaniem komend autostartu oczekuje 5 sekund na dowolny znak z portu szeregowego w celu umożliwienia przerwania ich wykonania. Zastosowanie interpretacji poszczególnych poleceń autostartu, umożliwia automatyczne wykonywanie dowolnych czynności znajdujących się na liście komend *bootloadera*, zapewniając ogromną elastyczność konfiguracji. W przypadku zewnętrznej pamięci Flash możemy wykonywać kod bezpośrednio z niej lub najpierw przekopiować kod z Flasha do RAM-u i dopiero z niego wykonywać. Wszystko zależy od potrzeb i wymagań aplikacji. W **tab. 1** przedstawiono listę komend interpretowanych przez *bootloader*.

Lista komend jest bardzo rozbudowana, ponieważ LBMON oprócz możliwości zaprogramowania i uruchomienia aplikacji/systemu operacyjnego z zewnętrznej pamięci, pełni rolę prostego monitora przydatnego podczas uruchamiania i diagnozowania urządzenia. Komenda *lram (adres)* umożliwia załadowanie programu do pamięci RAM mikrokontrolera. Jako parametr podajemy adres docelowy w pamięci. Program powinien być wysłany do portu szeregowego w postaci zakodowanej UUENCODE. Jest to bardzo prosty sposób kodowania, który zamienia dane 8–bitowe na 7–bitowe znaki ASCII. Do zamiany binarnego pliku na postać UUENCODE możemy użyć programu uuencode wpisując w wierszu polecenia: `uuencode sciezka_do_pliku.bin nazwa.bin > nazwa.uue`. Plik wynikowy w postaci uuencode wklejamy do terminala po wydaniu komendy *lram*. Kolejnym bardzo użytecznym poleceniem jest *pflash*, które umożliwia zaprogramowanie zewnętrznej pamięci Flash. Jako pierwszy argument polecenia podajemy heksadecymalnie adres względny początku pamięci Flash, który chcemy zaprogramować. Drugi argument stanowi adres początku obszaru pamięci RAM, natomiast trzeci argument określa wielkość obszaru do przekopiowania. Przed zapisaniem obszaru pamięci Flash musi on być wcześniej skasowany, co jest dokony-



Rys. 3. Algorytm działania *bootloadera*

wane automatycznie przez polecenie *pflash*. Należy pamiętać, że w pamięci Flash mogą być kasowane tylko całe sektory, tak więc kasowany obszar jest większy lub równy zapisywanemu obszarowi. Na przykład, jeżeli chcielibyśmy przepisać 2 kB pamięci RAM na początek pamięci Flash, to wywołujemy polecenie z następującymi argumentami: `pflash 0x0 0x81000000 2048`. Kopiowanie w odwrotną stronę z pamięci Flash do pamięci RAM umożliwia polecenie *copy*, którego pierwszy argument określa początek obszaru docelowego w pamięci RAM, drugi argument wskazuje na początek kopiowanego obszaru pamięci Flash w postaci względnej, natomiast trzeci argument określa wielkość kopiowanego obszaru. Polecenie to będzie szczególnie użyteczne podczas wykonywania kodu programu z pamięci RAM, która z reguły jest dużo szybsza niż pamięć Flash. Wykorzystując tę komendę w autostarcie (*autoboot*) możemy skopiować program z obszaru pamięci Flash do RAM–u, a następnie uruchomić go z pełną prędkością. Polecenie *go* umożliwia rozpoczęcie wykonywania kodu programu znajdującego się w dowolnym obszarze pamięci mikrokontrolera. Jako argument podajemy adres obszaru pamięci, który chcemy wykonywać. Należy pamiętać, że wektory wyjątków, niezależnie od wskazanego adresu, rozpoczynają się zawsze pod adresem 0x80000000, czyli na początku zewnętrznej pamięci Flash. Jeżeli chcemy uruchomić program w pamięci RAM, na początku Flasha, musimy umieścić krótki kod, przekierowujący wyjątki na początek RAM–u. W urządzeniach mogą być zastosowane pamięci o różnym czasie dostępu, dlatego po starcie *bootloadera* banki pamięci Flash i RAM są konfigurowane na maksymalną liczbę cykli oczekiwania, a po starcie *bootloadera* użytkownik może sam w zależności od szybkości zamontowanej pamięci skonfigurować poszczególne banki za pomocą polecenia *bcfg*. Polecenie *bcfg* może przyjmować 3 lub 5 argumentów. Pierwszy argument określa liczbę cykli oczekiwania, jakie procesor wykona podczas przełączania się pomiędzy bankami lub pomiędzy przejściem z trybu zapisu do odczytu i odwrotnie. Drugi i trzeci argument określa odpowiednio cykle oczekiwania dla zapisu i odczytu. Opcjonalny, czwarty argument jest odpowiednikiem 10. bitu w rejestrze BCFG i określa sposób używania linii BLS oraz WR. Ostatni argument umożliwia ustawienie szerokości magistrali. Również często zachodzi potrzeba wstępnego skonfigurowania portów GPIO przed uruchomieniem programu/systemu, co umożliwia komenda *gpio*. Jako pierwszy argument możemy podać numer określający port 0 lub 1, pozostałe porty nie są dostępne, ponieważ pełnią rolę magistrali. Drugi argument określa rodzaj operacji gdzie: *o* oznacza ustawienie odpowiednich linii portu w kierunku wyjścia, *i* ustawienie linii w kierunku wejścia, *s* ustawienie linii w stan 1, *c* ustawienie linii w stan 0. Wpisanie polecenia bez argumentów odczytuje stan linii portu.

Tab. 1. Lista komend interpretowanych przez bootloader

Komenda	Argumenty	Opis
ver	–	Wyświetla wersję oprogramowania bootloadera
testram	–	Testuje poprawność działania pamięci RAM. Wyświetla komunikat Test: Failed Test: PASSED
lram	adres_hex – obszar znajdujący się w pamięci RAM	Zapisuje dane przekazane pod adres wskazany w pamięci RAM. Dane powinny być wysyłane w postaci UUENCODE z terminala
plash	Flash _addr – adres względny w pamięci Flash ram_addr – adres obszaru pamięci RAM length – rozmiar obszaru	Kopiuje dane z pamięci RAM do pamięci Flash
dump	adres_hex – adres obszaru pamięci RAM lub Flash	Wyświetla zawartość pamięci Flash lub RAM o adresie przekazanym jako argument
go	adres_hex – adres obszaru pamięci Flash lub RAM	Rozpoczyna wykonywanie programu spod adresu wskazanego jako argument. Uwaga: wektory wyjątków zawsze rozpoczynają się w obszarze pamięci Flash (0x80000000)
autoboot	[brak] – wyświetla listę startową clear – czyści listę poleceń automatycznego startu cmd1; cmd2; cmd3 – Ustawia listę poleceń automatycznego startu	Pozwala ustawić, wyświetlić lub skasować listę poleceń automatycznego startu
copy	ram_addr – adres obszaru pamięci RAM Flash _addr – adres względny w pamięci Flash Length – rozmiar obszaru	Polecenie pozwala skopiować dane z pamięci Flash do pamięci RAM
write	b/w/d – 8bit/16bit/32bit hexadr – adres pamięci RAM hexval – wartość do zapisania	Zapisuje wartość pod adres w pamięci RAM
compare	adr1 – obszar 1 adr2 – obszar 2 len – rozmiar	Porównuje dwa obszary pamięci o określonym rozmiarze
fill	b/w/d – 8bit/16bit/32bit hexadr – adres w pamięci RAM val – wartość szesnastkowo len – liczba słów do zapisania	Wypełnia określony obszar pamięci RAM określoną wartością
bcfg	bank – numer banku pamięci idcy – liczba cykli pomiędzy RD/WR (0...15) rd – liczba cykli podczas odczytu (0...31) wr – liczba cykli podczas zapisu (0...31) [rble] – steruje zachowaniem linii BLS i WR 0 – wyłącza użycie linii BLS 1– włącza użycie linii BLS [bus] – szerokość magistrali 0 – 8 bitów 1 – 16 bitów 2 – 32 bity	Komenda konfiguruje czasy dostępu w bankach pamięci
gpio	port – numer portu (0 lub 1) s/c/i/o – ustaw 1/ustaw 0/wejście/wyjście hexval – maska bitowa linii portu	Komenda pozwala skonfigurować poszczególne linie GPIO przed uruchomieniem programu głównego.
help lub ?	–	Wyświetla krótki tekst pomocy

Używanie *bootloadera* LBMON

Po krótkim opisie komend objaśnimy w praktyce jak używać *bootloadera* na przykładzie wgrywania i uruchamiania jądra uCLinuxa. Równie dobrze, zamiast jądra uCLinuxa możemy uruchomić dowolny program lub system operacyjny. Pierwszą czynno-

ścią jaką musimy zrobić, aby używać *bootloadera* jest jego wgranie do wewnętrznej pamięci Flash mikrokontrolera. Możemy to wykonać za pomocą standardowego programu dostarczanego przez producenta mikrokontrolera LPC 2000 Flash Utility. W tym celu, w oknie wyboru pliku do zaprogramowania wybieramy plik lbmon.hex, konfigurujemy program, a następnie naciskamy przycisk *Upload to Flash*. Po zaprogramowaniu mikrokontrolera uruchamiamy terminal (np. Hyperterminal) z szybkością transmisji 115200, 1 bitem startu, 1 bitem stopu, bez kontroli przepływu i zerujemy mikrokontroler. Na terminalu powinien zgłosić się *bootloader* informując nas o typie oraz rozmiarze dostępnej pamięci:

```
**********************************
* LBLO LPC22xx BOOTLOADER v 1.12  *
*    Copyright L. Bryndza - EP    *
* email: lucjan.bryndza@ep.com.pl *
**********************************
SRAM test: PASSED
Flash type: STMicro - M29W320DB
SRAM at address:   0x81000000 Size:
4096kB
Flash at address:  0x80000000 Size:
4096kB

LBMON>
```

Gdy już upewnimy się, że LBMON pracuje poprawnie, przystępujemy do dalszych czynności związanych z wgraniem i uruchomieniem systemu. W tym przypadku kod jądra uCLinuxa będzie przechowywany w zewnętrznej pamięci Flash, natomiast przed uruchomieniem będzie kopiowany do pamięci RAM i stamtąd wykonywany. Jak już wcześniej wspomniano, na początku pamięci Flash trzeba umieścić prosty kod przedstawiony na **list. 1** przekierowujący wektory wyjątków z pamięci Flash do RAM.

Dokonujemy tego wgrywając plik `wektory.uue` do pamięci RAM. W tym celu z terminala wydajemy komendę `lram 0x81000000`, na ekranie powinien pojawić się komunikat zachęcający do przesłania pliku w postaci `uuencode`:

```
OK please send file to host.
```

Teraz wklejamy do terminala plik wektory.uue. W przypadku Hyperterminala, z menu wybieramy „Wklej plik tekstowy do Hosta”. Po zakończeniu przesyłania pliku, *loader* wyświetli jego rozmiar oraz sumę kontrolną CRC32:

```
OK transfer complette: 64 CRC32:
ECB7E7F1
```

Po załadowaniu programu do pamięci RAM programujemy pamięć Flash wpisując komendę: `pflash 0x0 0x81000000 64`, czyli przepisujemy 64 bajty z początku pamięci RAM do pamięci Flash. Po zakończeniu programowania pamięci Flash na terminalu powinniśmy zobaczyć:

List. 1. Kod przekierowujący wektory wyjątków z pamięci Flash do RAM

```
Vectors:        LDR     PC, Reset_Addr
                LDR     PC, Undef_Addr
                LDR     PC, SWI_Addr
                LDR     PC, PAbt_Addr
                LDR     PC, DAbt_Addr
                NOP                     /* Reserved Vector */
                LDR     PC, IRQ_Addr
                LDR     PC, FIQ_Addr

Reset_Addr:     .word   0x81008000
Undef_Addr:     .word   0x81000004
SWI_Addr:       .word   0x81000008
PAbt_Addr:      .word   0x8100000C
DAbt_Addr:      .word   0x81000010
                .word   0               /* Reserved Address */
IRQ_Addr:       .word   0x81000018
FIQ_Addr:       .word   0x8100001C
```

```
Erasing range (0x000000 - 0x003FFF)
Erasing... - DONE
Programming range (0x000000
- 0x00003F)
Programming... - DONE
Verifying... - DONE
OK Flash write succesfull
```

Mamy już zaprogramowany początek pamięci Flash przekierowujący wektory wyjątków do obszaru RAM, możemy więc przystąpić do wgrania właściwego pliku z kodem jądra uCLinuxa *Linux.bin*. Kod jądra uCLinuxa będziemy przechowywać w pamięci Flash za wektorami wyjątków pod adresem 0x8000. W celu przesłania pliku kodujemy plik *Linux.bin* do postaci `uuencode` wydając w wierszu poleceń systemu operacyjnego polecenie: `uuencode Linux.bin Linux.bin > Linux.uue`. Jeżeli ktoś nie chce korzystać z wiersza polecenia (99% użytkowników Windows), do zakodowania pliku do postaci `uuencode` może wykorzystać bardzo popularny program „Total Commander”, gdzie w menu Plik znajduje się polecenie *Zakoduj plik (MIME,UUE,XXE)* Następnie w terminalu wpisujemy: `lram 0x81008000` i tak jak poprzednio przesyłamy plik poprzez port szeregowy. W zależności od rozmiaru programu przesyłany plik może się wgrywać dość długo, a podczas przesyłania pliku każdą odebraną linię LBMON potwierdza znakiem „.” Po wgraniu pliku, *loader* wyświetli informację o rozmiarze i sumie CRC32 odebranego pliku:

```
OK transfer complette: 1068976
CRC32: E8E63264.
```

Pozostało nam jeszcze przepisanie programu z pamięci RAM do Flash, czego dokonujemy wydając w terminalu polecenie: `pflash 0x8000 0x81008000 1068976`. Po prawidłowym zaprogramowaniu pliku otrzymamy informację na terminalu:

```
Erasing range (0x008000 - 0x10FFFF)
Erasing... - DONE
Programming range (0x008000 - 0x10C-
FAF)
Programming... - DONE
Verifying... - DONE
OK Flash write succesfull
```

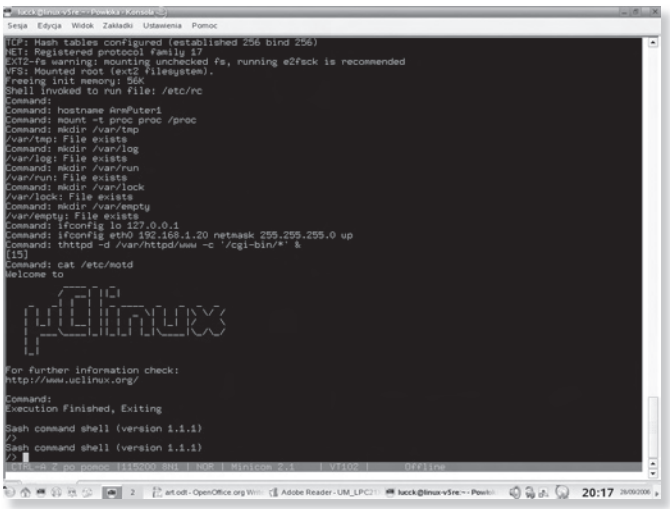
Ostatnią czynnością jest ustawienie autostartu tak, aby wgrany program uruchamiał się automatycznie po włączeniu zasilania oraz ustawieniu cykli oczekiwania w zależności od posiadanej pamięci RAM i Flash. Przed uruchomieniem możemy także ustawić wymagane linie portów GPIO. Dalszy ciąg opisu należy tutaj traktować jako przykład, gdyż płytka jaką dysponujemy może posiadać inne urządzenia peryferyjne czy pamięci o innym czasie dostępu. Czynnościami jakich musimy dokonać przed startem jądra systemu to:

- skonfigurowanie banku pamięci RAM na 4 cykle oczekiwania pomiędzy zmianami banku, i 3 cykli dla zapisu i odczytu – komenda: `bcfg 1 4 3 3`
- skonfigurowanie banku pamięci Flash na 4 cykle oczekiwania pomiędzy zmianami banku i 4 cykle oczekiwania dla zapisu i odczytu – komenda: `bcfg 0 4 4 4`
- skonfigurowanie banku B2 (0x82000000), do którego jest podłączona karta sieciowa RTL8019 w trybie 16–bitowym na 7 cykli oczekiwania – komenda: `bcfg 2 6 6 6 1 1`

Ustawienie linii P0.23 RESET układu w stan niski, tak aby uaktywnić sygnał RESET karty sieciowej RTL8019 – komendy: `gpio 0 o 0x00800000` oraz `gpio 0 c 0x00800000`

- skopiowanie z pamięci Flash spod adresu 0x8000 kodu Linuksa w obszar pamięci RAM 0x81008000 – komenda: `copy 0x81008000 0x8000 1068976`
- rozpoczęcie wykonywania jądra Linuksa od adresu 0x81008000 komenda: `go 0x81008000`.

Wszystkie wymienione komendy wpisujemy do autostartu za pomo-



Rys. 4. Okno konsoli uCLinuxa

cą polecenia *autoboot* oddzielone za pomocą średników. Czyli w naszym przypadku:

```
autoboot bcfg 1 3 3 4;
gbcfg 2 6 6 6 1 1;
gpio 1 o 0x00800000; gpio
1 c 0x00800000; copy
0x81008000 0x8000 1200000;
go 0x81008000
```

Po ustawieniu poleceń startowych i zrestartowaniu urządzenia, *bootloader* po 5 sekundach przejdzie do wykonywania wszystkich wprowadzonych komend, a w efekcie do uruchomienia programu/systemu. Na **rys. 4** przedstawiono uruchomiony za pomocą naszego *bootloadera* system uCLinux.

W przedstawionym powyżej przykładzie, przed startem programu głównego mikrokontroler przechodził do *bootloadera*, który konfigurował i kopiował nasz program do pamięci RAM oraz go uruchamiał. Istnieje również możliwość uruchomienia załadowanego programu bez pośrednictwa *bootoloadera* wprost z zewnętrznej pamięci Flash. Jednak program musi wówczas samodzielnie konfigurować kontroler pamięci zewnętrznych (EMC). Start systemu bezpośrednio z zewnętrznej pamięci Flash jest możliwy poprzez ustawienie linii BOOT0 w stan wysoki oraz linii BOOT1 w stan niski.

Coś dla leniwych, czyli program dla Win32

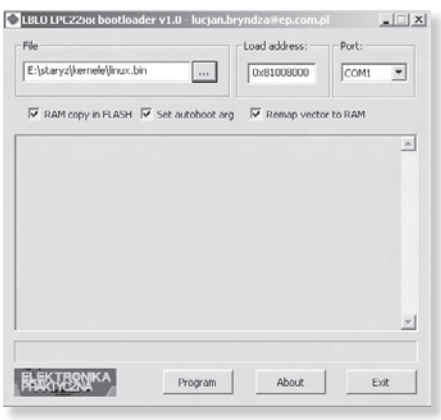
Dla użytkowników, którzy boją się pracy w trybie tekstowym (czytaj użytkowników systemu MS Windows), mam dobrą wiadomość, mianowicie został przygotowany program *LBLO Loader* umożliwiający wgranie programu do urządzenia docelowego za pomocą kilku kliknięć myszką. Jedyną czynnością jaką możemy zrobić po wgraniu programu, to ewentualnie modyfikacja komend autostartu, tak aby dostosować ją do pamięci i sprzętu zainstalowanego w naszej płytce. Okno programu LBLO Manager przedstawiono na **rys. 5**.

W polu „*File*" możemy wybrać plik do zaprogramowania pamięci zewnętrznej. Program umożliwia załadowanie plików w postaci binarnej, jak i w formacie IntelHex. W przypadku, gdy ładujemy plik binarny, wówczas pole „*Load Address*" określa adres w pamięci, gdzie plik zostanie przesłany, natomiast gdy przesyłamy plik HEX, wówczas pole to jest ignorowane, ponieważ format HEX zawiera informację o docelowym adresie w pamięci. Pole „*Port*" umożliwia wybranie portu szeregowego. Szerszego komentarza wymagają opcje dodatkowe. Pierwsze pole „*RAM copy in Flash*" w przypadku, gdy adres pamięci znajduje się w obszarze pamięci RAM umożliwia ustawienie autostartu, tak aby po włączeniu płytki program był kopiowany z pamięci Flash do RAM. W przypadku wybrania tej opcji, program jest kopiowany do pamięci Flash od adresu 0x8000. Zaznaczenie pola „*Set autoboot arg*" powoduje ustawienie komend wykonywanych automatycznie (polecenie *autoboot*). Zaznaczenie pola „*Remap vector to RAM*", gdy program jest umieszczony w pamięci RAM powoduje zapisanie na początku pamięci Flash wspomnianego wcześniej kodu przekazującego wektory do pamięci RAM. Poniżej przycisku opcji znajduje się okno tekstowe wyświetlające przebieg procesu programowania. Poniżej tego pola znajduje się pasek informujący o postępie przesyłania pliku. Na samym dole znajdują się przyciski umożliwiające rozpoczęcie procesu programowania, przycisk wyświetlający informację o programie oraz przycisk wyjścia z programu. Aby wgrać do urządzenia docelowego wspomniane wcześniej jądro uCLinuxa, w menu *File* wybieramy plik *Linux.bin*. W polu „*Load Addres*" wpisujemy adres ładowania kodu programu na 0x81008000 (obszar pamięci RAM) oraz zaznaczamy wszystkie opcje dodatkowe. Następnie wciskamy przycisk „*Program*" i czekamy na zakończenie procesu programowania. Gdy zaprogramowany kod sam nie modyfikuje ustawień kontrolera pamięci EMC, lub gdy konieczna jest inicjalizacja portów GPIO przed uruchomieniem programu/systemu, należy za pomocą terminala (komenda *autoboot*) ustawić wymagane polecenia (*bcfg* lub *gpio*).

Zakończenie

Zaprezentowany tutaj *Bootloader* zapewnia podstawową funkcjonalność wymaganą w tego typu programach. Umożliwia wstępne skonfigurowanie urządzeń peryferyjnych przed startem systemu głównego, zaprogramowanie zewnętrznej pamięci Flash oraz przekopiowanie programu z pamięci Flash do RAM w celu przyśpieszenia wykonywania programu. Jest również bardzo pomocny podczas uruchamiania sprzętu. Został on napisany tak, aby konfiguracja i programowanie odbywały się za pomocą poleceń tekstowych z wykorzystaniem terminala. Trochę to komplikuje cały program, ale ma tę zaletę, że jest niezależne od systemu operacyjnego. Na pewno zauważą to dyskryminowani zazwyczaj użytkownicy Linuksa. Wadą LBMON jest stosunkowo powolny transfer danych za pomocą interfejsu RS232. Jednak w przypadku, gdy nasze urządzenie posiada interfejs sieciowy możemy pokusić się o rozbudowanie *bootloadera* o możliwość przesyłania danych za pomocą sieci Ethernet. Istnieje również możliwość bardzo łatwego dopisywania kolejnych poleceń do *bootloadera*, umożliwiając jego dalszą rozbudowę.

Lucjan Bryndza, EP
lucjan.bryndza@ep.com.pl



Rys. 5. Okno programu LBLO Manager