Przedmowa

Proponując Czytelnikowi kolejną na naszym rynku książkę o mikrokontrolerach, powinienem na wstępie udzielić odpowiedzi na dwa istotne pytania. O czym jest ta książka? Do kogo jest skierowana?

Na pierwsze pytanie najkrócej mógłbym odpowiedzieć, że książka traktuje o implementowaniu protokołów i tworzeniu aplikacji sieciowych na mikrokontrolery, które są wyposażone w interfejs Ethernet. Wydajność obliczeniowa mikrokontrolerów stale rośnie (przy zachowaniu poboru mocy w granicach rozsądnych dla tego rodzaju układów). Umożliwia to zaimplementowanie stosu protokołów sieciowych TCP/IP, dotychczas dostępnego tylko na dużych maszynach i komputerach osobistych. Moim zdaniem spowoduje to rewolucję zastosowań mikrokontrolerów, podobną do tej, jaką spowodowało kilkadziesiąt lat temu łączenie komputerów osobistych w sieci lokalne (ang. *local area networks*) i intersieci (ang. *internets*). Urządzenia sterowane mikrokontrolerami mogą stać się częścią Internetu, co stwarza zupełnie nowe możliwości.

Specyfika urządzeń sterowanych mikrokontrolerami polega między innymi na konieczności reagowania na wiele zdarzeń zewnętrznych i sterowania wieloma peryferiami. Czasy reakcji powinny być na tyle krótkie, aby sprawiać wrażenie, że wszystko dzieje się niejako równolegle. Stosowane są tu dwa podejścia:

– cała aplikacja napisana jest w postaci jednego programu, mającego dostęp do całości zasobów sprzętowych, bezpośrednio obsługującego układy peryferyjne i wykorzystującego intensywnie system przerwań
– działaniem mikrokontrolera steruje system operacyjny, ukrywający obsługę sprzętu i przerwań w sterownikach urządzeń, a aplikacja jest podzielona na zadania, procesy lub wątki wykonywane współbieżnie pod kontrolą tego systemu operacyjnego.

W niniejszej książce prezentuję pierwsze podejście, które sprzyja zmniejszaniu rozmiaru pamięci zajmowanej przez aplikację, co bywa nadal istotne w systemach wbudowanych.

Książka ta ma w założeniu być podręcznikiem i przewodnikiem. Chcąc zrealizować zasadę, że najlepiej uczyć się na przykładach, stanąłem przed koniecznością wyboru platformy sprzętowej i środowiska programistycznego. Prezentowane w książce zagadnienia ilustruję przykładami napisanymi na mikrokontroler STM32F107 z rdzeniem ARM Cortex-M3 firmy STMicroelectronics. Do ich przetestowania użyłem zestawu ZL29ARM z modułem ethernetowym ZL3ETH oraz wyświetlaczem ciekłokrystalicznym WG12864A. Przykłady napisane są w języku C, w którym obecnie najczęściej programuje się mikrokontrolery. Kompilowałem je za pomocą GCC w wersji 4.4.3, z wykorzystaniem GNU Binutils w wersji 2.20.1 i Newlib w wersji 1.18. Użyłem w nich publicznie dostępnej implementacji stosu TCP/IP – biblioteki lwIP w wersji 1.3.2. Skorzystałem też z bibliotek dostarczanych przez STMicroelectronics: CMSIS (*Cortex Microcontroller Software Interface Standard*) w wersji 1.30, STM32F10x Standard Peripherals Library w wersji 3.3.0 i STM32 ETH Firmware Library w wersji 1.1.0.

Świadomie zrezygnowałem z pisania przykładów w języku C++, który zyskuje ostatnio coraz większą popularność w świecie mikrokontrolerów. C++ ujawnia swoje zalety przy dużych programach, zwłaszcza gdy intensywnie korzysta się ze standardowej biblioteki wzorców STL (ang. *Standard Template Library*). Nie ma co ukrywać, że C++ jest bardzo trudnym językiem. Chcę, aby książka była w pełni dostępna dla Czytelnika nieznającego C++. Używając pełnej siły wyrazu C++, ryzykowałbym niezrozumienie przykładów, a nie ma tu miejsca na szczegółowe objaśnianie zaawansowanych konstrukcji tego języka. Natomiast pisanie przykładów w jakiejś okrojonej wersji C++ byłoby tylko niepotrzebnym dodawaniem lukru obiektowości. Czytelnik znający C++ bez problemu przełoży na ten język przykłady napisane w C.

Przykłady starałem się pisać w sposób ogólny, aby mogły być łatwo przeniesione na inne mikrokontrolery. Jedynym rzeczywiście specyficznym i mocno zależnym od sprzętu fragmentem kodu jest sterownik (ang. *driver*) pośredniczący między biblioteką lwIP a mikrokontrolerem. Szczegółowo wyjaśniam, jak napisać taki sterownik i jak dopasować bibliotekę lwIP do architektury konkretnego mikrokontrolera. Pozostała część prezentowanego przeze mnie oprogramowania wcale lub tylko w bardzo niewielkim stopniu zależy od architektury sprzętu i daje się łatwo zaadaptować na dowolny mikrokontroler.

Postaram się teraz odpowiedzieć na drugie z postawionych na początku pytań. Książka przeznaczona jest dla osób, które już mają jakieś doświadczenie w programowaniu mikrokontrolerów. Zakładam, że Czytelnik umie programować w C. Chcącym poszerzyć swoją wiedzę o tym języku polecam klasyczną książkę *Język ANSI C*, napisaną przez jego twórców, Kernighana i Ritchiego [2]. Nie jest potrzebna znajomość Asemblera ARM. Oczekuję co najmniej podstawowej wiedzy z zakresu elektroniki, tak aby na podstawie opisów i schematów zamieszczonych w książce oraz danych katalogowych umieć samodzielnie zmontować układ prototypowy lub wykonać niezbędne połączenia w jakimś zestawie uruchomieniowym, których wiele jest dostępnych na rynku. Nie oczekuję natomiast żadnej wiedzy o sieciach komputerowych czy protokołach sieciowych – staram się wyjaśniać wszystko od podstaw. Zakładam też, że Czytelnik ma wystarczające obycie informatyczne, aby samodzielnie zainstalować środowisko programistyczne (edytor, kompilator itd.) i zapoznać się ze szczegółami jego obsługi. Mam nadzieję, że książka będzie ciekawą lekturą zarówno dla profesjonalisty, zawodowo projektującego układy z mikrokontrolerami, jak i hobbysty zajmującego się tym tylko dla własnej satysfakcji. Może też zainteresować osoby, które nigdy dotąd nie programowały mikrokontrolerów, a chciałyby się dowiedzieć, jak implementuje się protokoły sieciowe na taki komputer.

Wydaje mi się, że dodatkowym atutem książki jest zaprezentowanie Czytelnikowi przedstawiciela nowoczesnej rodziny mikrokontrolerów STM32. Jest to niewątpliwie rodzina układów, które mają szansę odnieść rynkowy sukces w najbliższej przyszłości. Wybrany do napisania przykładów mikrokontroler STM32F107 należy do linii zorientowanej na komunikację (ang. *connectivity line*). Oprócz zwykle spotykanych w mikrokontrolerach interfejsów I^2C, SPI i USART, wyposażony jest w CAN, Ethernet, I^2S i USB. Interfejs Ethernet może pracować z przepływnością 10 Mb/s lub 100 Mb/s w trybie dwukierunkowym naprzemiennym (ang. *half-duplex*) lub dwukierunkowym jednoczesnym (ang. *full-duplex*). Interfejs USB może pracować

z przepływnością 12 Mb/s (ang. *full speed*) jako urządzenie (ang. *device*) oraz z przepływnością 1,5 Mb/s (ang. *low speed*) lub 12 Mb/s jako host. Ponadto STM32F107 zawiera dość typowe układy licznikowe oraz przetworniki analogowo-cyfrowe i cyfrowo-analogowe. Skonstruowany jest w oparciu o 32-bitowy rdzeń ARM Cortex-M3, który można taktować maksymalnie z częstotliwością 72 MHz. Rdzeń ten zaprojektowano specjalnie do sterowania systemami głęboko wbudowanymi (ang. *deeply embedded*). Oferuje rozbudowany system przerwań z krótkim i przewidywalnym czasem rozpoczęcia obsługi oraz typowe dla tego segmentu układów tryby oszczędzania energii. Rdzeń obsługuje tylko zestaw instrukcji Thumb-2, rozszerzający zestaw instrukcji Thumb o kodach 16-bitowych o instrukcje o kodach 32-bitowych. Thumb-2 jest reklamowany jako osiągający wydajność tylko nieznacznie mniejszą od zestawu instrukcji ARM o kodach 32-bitowych przy zmniejszeniu o około 25% zapotrzebowania na pamięć programu, co jest porównywalne z gęstością kodu osiąganą przy zastosowaniu okrojonego zestawu instrukcji Thumb, który jednak cechuje się istotnie mniejszą wydajnością od pełnego zestawu instrukcji ARM.

Układ książki jest następujący. Rozdział 1 pełni funkcję rozbudowanego wstępu. Poruszam w nim zagadnienia niezwiązane bezpośrednio z tytułem książki, ale potrzebne w dalszej jej części. Opisuję obsługę wejść i wyjść ogólnego przeznaczenia (ang. *general purpose input-output*) na przykładzie diod świecących i wyświetlacza ciekłokrystalicznego. Wyjaśniam strukturę archiwum z przykładami i parametry kompilacji. Omawiam też organizację pamięci programu oraz działanie procedury startowej (ang. *startup code*) i skryptu konsolidatora (ang. *linker script*). Rozdział 2 zaczynam od przedstawienia modelu warstwowego intersieci. Następnie opisuję technologię Ethernet, ale skupiam się tylko na jej wariantach i trybach pracy obsługiwanych przez mikrokontroler. Przedstawiam budowę ramki i rodzaje adresów ethernetowych. Następnie wyjaśniam, jak ramki podstawowego protokołu intersieci, czyli IP (ang. *Internet Protocol*), są przesyłane w ramkach Ethernet. Opisują też działanie ARP (ang. *Address Resolution Protocol*), czyli protokołu tłumaczącego adresy IP na adresy ethernetowe. Rozdział ten kończę przykładem prostego monitora sieci. W rozdziałach 3 i 4 omawiam bibliotekę lwIP implementującą stos protokołów TCP/IP. W rozdziale 3 zajmuję się współpracą tej biblioteki ze sprzętem – pokazuję trzy wersje sterownika Ethernetu. Poszczególne sterowniki różnią się trybem pracy DMA (ang. *Direct Memory Access*) oraz sposobem obsługi buforów odbiorczych i nadawczych. Pokazuję wersję z kopiowaniem zawartości buforów i bez kopiowania ich zawartości (ang. *zero copy*). Na końcu tego rozdziału przedstawiam nieco informacji o protokołach ICMP (ang. *Internet Control Message Protocol*) i DHCP (ang. *Dynamic Host Configuration Protocol*). W rozdziale 4 zajmuję się interfejsem programistycznym API (ang. *application programming interface*) między biblioteką lwIP a aplikacją. Opisuję podstawowy interfejs udostępniany przez tę bibliotekę, czyli interfejs surowy (ang. *raw*). Pokazuję, jak wykorzystać ten interfejs do tworzenia aplikacji w modelu klient-serwer. W rozdziale tym przedstawiam też niezbędne informacje o protokołach TCP (ang. *Transmission Control Protocol*) i UDP (ang. *User Datagram Protocol*). Rozdział 4 jest jedynym w książce niezakończonym żadnym przykładem, gdyż właściwie kolejne rozdziały do końca książki zawierają przykłady korzystające z wiadomości przedstawionych w tym rozdziale. W rozdziale 5 opisuję dwa przykłady serwerów TCP, czyli podstawowego strumieniowego pro-

tokołu warstwy transportowej intersieci. W drugim z tych serwerów uruchamiam układ nadzorcy (ang. *watchdog*) w celu zopobieżenia trwałemu zawieszeniu się programu. W rozdziale 6 omawiam dwa przykłady klientów TCP. Korzystają one z wbudowanego w mikrokontroler zegara czasu rzeczywistego RTC (ang. *Real Time Clock*), w celu uruchamiania się w zadanych odstępach czasu. Ponadto korzystają też z trybu czuwania (ang. *standby*), aby zmniejszyć pobór prądu zasilania między kolejnymi uruchomieniami. Drugi z klientów dodatkowo używa rejestrów zapasowych (ang. *backup registers*) mikrokontrolera STM32F107, aby między kolejnymi uruchomieniami przechować w nich dane aplikacji. W rozdziale 7 zamieszczam przykład prostego serwera UDP, czyli podstawowego pakietowego protokołu warstwy transportowej intersieci. Serwer ten pozwala zdalnie sterować portami wejścia-wyjścia mikrokontrolera. W rozdziale 8 opisuję przykład klientów dwóch protokołów bazujących na UDP, a mianowicie DNS (ang. *Domain Name System*) i SNTP (ang. *Simple Network Time Protocol*). SNTP służy do synchronizacji czasu w sieci i umożliwia zsynchronizowanie zegara czasu rzeczywistego mikrokontrolera z jakimś dostępnym w Internecie atomowym wzorcem czasu. W rozdziale 9 zajmuję się rozgłaszaniem (ang. *broadcast*) w UDP. W rozdziale 10 pokazuję, jak skomunikować się z mikrokontrolerem za pomocą przeglądarki internetowej i opisuję implementację prostego serwisu WWW (ang. *World Wide Web*). Listę książek i dokumentów, z którymi warto się zapoznać, zamieszczam w spisie literatury. W dodatku opisuję narzędzia GNU, których użyłem do skompilowania przykładów.

Duża część książki, począwszy od rozdziału 4, zawiera szczegółowe opisy, jak implementuje się własne protokoły i tworzy aplikacje korzystające bezpośrednio z protokołów transportowych TCP i UDP. Wszystkie przykłady używają interfejsu surowego, który pozwala na lepszą integrację aplikacji ze stosem TCP/IP. Interfejs surowy jest preferowany w systemach wbudowanych, gdyż ma małe wymagania pamięciowe i nie potrzebuje dużej mocy obliczeniowej – nie ma wątków ani procesów, nie trzeba zmieniać kontekstu. Przykładowe programy są sterowane zdarzeniami (ang. *event based*) za pomocą funkcji zwrotnych (ang. *callback functions*). Źródłem zdarzeń są przerwania. Przykłady konstruuję przyrostowo – kolejne korzystają z modułów zaprogramowanych na potrzeby poprzednich przykładów.

Wszystkie pliki źródłowe przykładów, łącznie ze źródłami biblioteki lwIP oraz źródłami specyficznych dla mikrokontrolerów STM32 bibliotek dostarczanych przez STMicroelectronics, dostępne są w postaci skompresowanego archiwum, które można ściągnąć ze strony Wydawnictwa BTC *http://www.btc.pl* lub strony domowej Autora *http://www.mimuw.edu.pl/~marpe/book*.

Chcę wyraźnie zaznaczyć, że książka nie jest instrukcją obsługi narzędzi programistycznych ani nie stanowi wyczerpującej dokumentacji sprzętu użytego do przetestowania zamieszczonych przykładów. Czytelnik z łatwością znajdzie potrzebną dokumentację na stronach internetowych odpowiednich producentów.

Staram się promować polską terminologię. Dla wygody Czytelnika, zwykle zmuszonego korzystać z dokumentacji w języku angielskim, przy pierwszym, a czasem i przy kolejnym, użyciu terminu, który nie jest powszechnie znany, umieszczam w nawiasach jego angielski odpowiednik.

Marcin Peczarski, Warszawa 2011