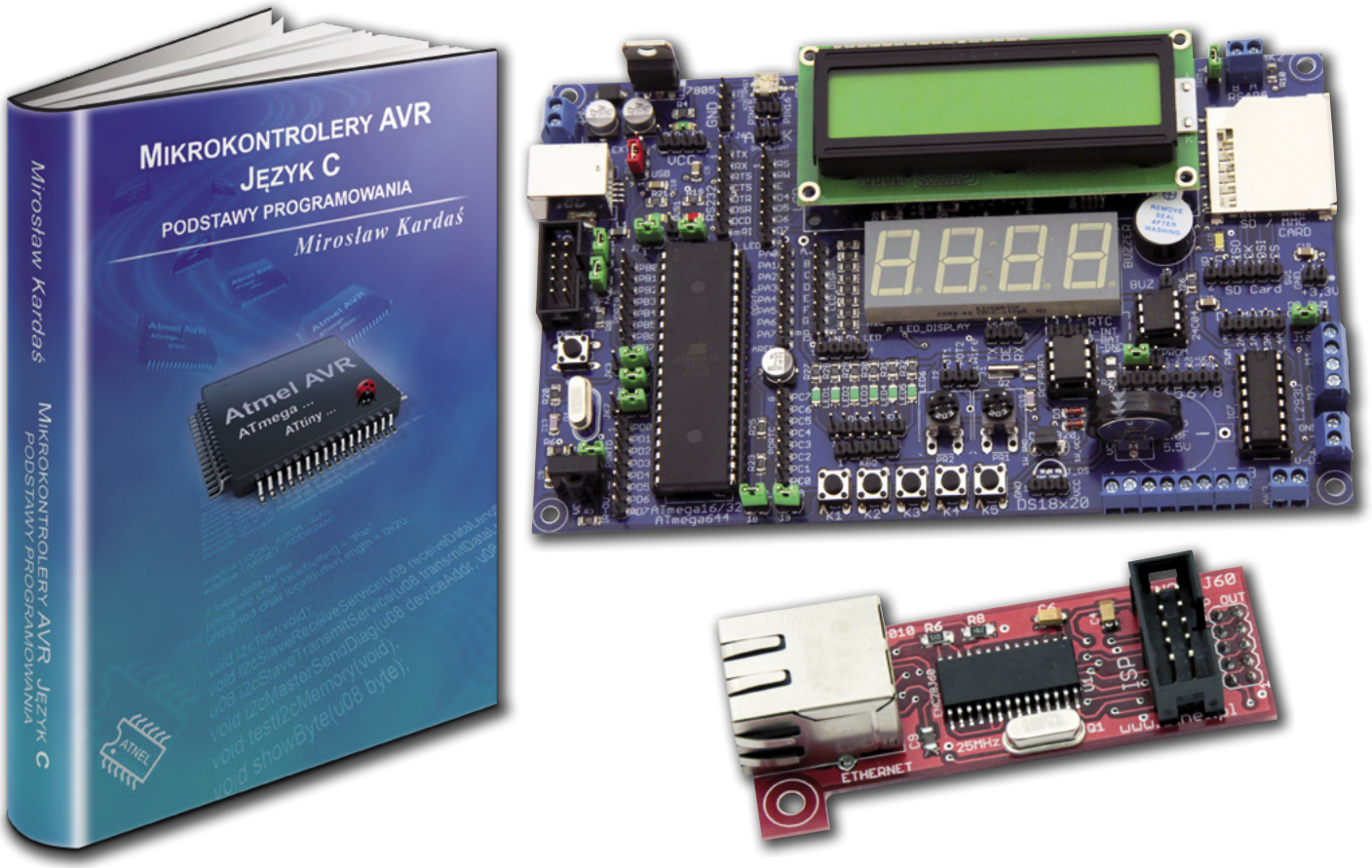


Mikrokontrolery AVR

Język C – podstawy programowania: serwer HTTP

Z końcem 2010 roku na polskim rynku pokazała się wydana przez firmę Atnel książka autorstwa Mirosława Kardasia pt: „Mikrokontrolery AVR – Język C – Podstawy Programowania". Postanowiliśmy przybliżyć czytelnikom tę interesującą pozycję poprzez opublikowanie jej fragmentu poświęconego tworzeniu serwera http na mikrokontrolerach AVR. Przykłady prezentowane w książce mogą być z łatwością wykonane z użyciem zestawów deweloperskich firmy Atnel.

Przykład realizacji serwera http został oparty na istniejących i ogólnodostępnych bibliotekach z Internetu. Myślę, że dla każdego początkującego dobrym rozwiązaniem będzie implementacja stosu TCP oferowana przez autorów z witryny internetowej: tuxgraphics.org. Udostępniają oni bezpłatnie własną implementację stosu dla niewielkich nawet mikrokontrolerów AVR, takich jak np. ATmega88, a na dodatek przedstawiają mnóstwo przykładów konkretnych zastosowań oraz form ich realizacji. Problemem dla wielu początkujących elektroników w Polsce może być jednak nieco słabsza znajomość języka angielskiego, co w połączeniu z brakiem podstawowych zasad działania i obsługi stosu TCP powoduje spore trudności nie tylko w uruchamianiu gotowych projektów, ale szczególnie podczas tworzenia własnych, w oparciu o przekazywane na stronie informacje. Dodatkowym utrudnieniem jest, że wszystkie przykłady komunikacji od strony PC przedstawiane tam są w oparciu o system operacyjny Linux a nie popularny Windows, co także może prowadzić do frustracji przy pierwszym niepowodzeniu podczas uruchamiania takich projektów. Postaram się zatem zwrócić uwagę na najbardziej istotne elementy podczas prób uruchamiania programów testowych, a także istotne z punktu wprowadzania własnych modyfikacji i stosowania rozwiązań już we własnych układach. Kod potrzebny do utworzenia podstawowego serwera, który będzie w odpowiedzi na zapytanie z przeglądarki wyświetli nieskomplikowaną stronę, jest niesamowicie prosty i krótki, z udziałem najnowszej wersji implementującej stos TCP ze strony tuxgraphics.org (rys. 1.)

W oparciu o tak prosty serwer z udziałem mikrokontrolera AVR można wyświetlać na własnej stronie www obrazki dowolnej wielkości. W tym konkretnym przypadku za-

Rys. 1. Przykład prostej strony www

stosowałem pewną sztuczkę. Polega ona na tym, że fizycznie obrazek nie jest przechowywany w pamięci mikrokontrolera. W kodzie HTML strony istnieje tylko odnośnik do obrazka znajdującego się na innym serwerze www. Wykorzystując zatem tak proste możliwości jesteśmy w stanie skonstruować urozmaiconą stronę, która docelowo będzie prezentować wyniki z naszych czujników. Wszystko to kwestia pomysłów. Przypomnę, że można także obrazki przechowywać w pamięci FLASH, jeśli są niewielkie i mamy odpowiednio dużo wolnego miejsca. Można także wykorzystać kartę pamięci SD do przechowywania plików .gif lub jpg. Trzeba się jednak liczyć z tym, że w przypadku odczytu strony należy zorganizować za pomocą magistrali SPI najpierw odczyt obrazka do bufora w pamięci RAM, a następnie przesłać go za pomocą SPI do układu ENC28J60. W tak małych mikrokontrolerach, jak ATmega88/168/328 czy ATmega32, a nawet ATmega644, mamy do dyspozycji stosunkowo mało pamięci RAM, dlatego czasami przesłanie jednego większego obrazka trzeba będzie jeszcze podzielić na kilka etapów. Jest to możliwe, jeśli uprzednio przygotujemy sobie binarne wzorce plików graficznych różnego typu. Pozostaje wtedy tylko kwestia czasu potrzebnego na przesłanie i jednoczesny dostęp do strony www z zewnątrz.

Przejdźmy do krótkiego omówienia samej biblioteki oraz naszego kodu. Autorzy omawianego stosu TCP w obecnej wersji postawili na maksymalne możliwe uproszczenie wykorzystania jego zasobów. Funkcje zostały odpowiednio pogrupowane w plikach, dzięki czemu jest ich niewiele. Ich lista została zawarta w tabeli 1.

Zmiany kosmetyczne

Na początek musimy się zabrać za wprowadzenie pewnych zmian kosmetycznych w niektórych plikach. Zmiany związane są z tym, że autorzy dostosowali swoje biblioteki tylko do mikrokontrolerów: ATmega88/168/328 oraz ATmega644. My natomiast chcemy rozszerzyć te możliwości dla ATmega16 oraz ATmega32. W związku z powyższym zmian dokonamy w pliku enc28j60.c. Musimy też usunąć dwukrotne wpisy z definicją F_CPU w plikach: enc28j60.c oraz timeout.h. Decydujemy się całkowicie na to, że w naszych projektach F_CPU będzie zawsze zdefiniowane w pliku „makefile" automatycznie generowanym przez środowisko Eclipse lub AVR Studio. Jeżeli ponadto będziemy korzystali zawsze z najnowszej wersji AVR GCC, praktycznie możemy pozbyć się dla kolejnego uproszczenia pliku timeout.h. Zamiast niego będziemy dołączać systemowy nagłówek: #include <util/delay.h>, tym bardziej, że być może sami będziemy korzystać z funkcji w nim zawartych. Kolejna poprawka, tym razem czysto kosmetyczna, polega na tym, iż możemy całkowicie wyłączyć kod odpowiedzialny za ustawianie sygnału taktującego na wyjściu CLKOUT układu ENC28J60. Jest to związane z tym, że wszystkie projekty ze strony tuxgraphics.org bazują na założeniu, że mikrokontrolery AVR taktowane są właśnie tym sygnałem z układu karty sieciowej. W rozwiązaniach proponowanych przeze mnie zmieniamy całkowicie taktykę. Nasza karta sieciowa oparta o układ ENC28J60 jest buforowana i oddzielnie zasilana, o czym już wcześniej wspominałem. Nie posiada wyprowadzonego sygnału CLKOUT, gdyż jest całkowicie niepotrzebny. Nasze mikrokontrolery mogą być taktowane zewnętrznym rezonatorem w granicach od 11,0592 MHz do 20 MHz. Zmiany będą dotyczyły pliku enc28j60.c: usuniemy jedną całą funkcję, której nie będzie trzeba wywoływać podczas inicjalizacji układu.

Zacznijmy od konfiguracji bibliotek do naszych potrzeb. W rzeczywistości na początku możemy skorzystać z jednego z gotowych przykładów serwera w pliku „basic_web_sever_example.c". Zanim będziemy go kompilować, musimy zdecydować, jakiego pinu będziemy używać do sygnału CS. Domyślnie biblioteki korzystają ze sprzętowego SPI oraz dedykowanych w tym celu pinów łącznie z wykorzystaniem domyślnego pinu oznaczonego w mikrokontrolerze jako SS. Trzeba mieć jednak tego świadomość, szczególnie gdy zaprzęgniemy w przyszłości do pracy także obsługę karty pamięci SD, która także będzie korzystać ze sprzętowego SPI. Wtedy trzeba rozdzielić na osobne piny sygnały CS dla każdego z tych modułów. Opisanych tutaj zmian można dokonać na początku pliku enc28j60.c. Sam umieściłbym ten fragment w pliku enc28j60.h, jednak pozostawię to w celu kompatybilności z przyszłymi wersjami tych bibliotek.

Widać (list. 1.), że piny odpowiadające za sprzętowe SPI są automatycznie definiowane w zależności od zastosowanego mikrokontrolera. Tutaj właśnie dodajemy naszą poprawkę dla umożliwienia pracy programów na ATmega16/32. Poszerzyłem sprawdzanie dodatkowych dwóch mikrokontrolerów w drugim warunku rozpoczynającym się od **#if defined(__AVR_ATmega644__)**. Widoczne także są miejsca do własnej redefinicji pinu CS w liniach: **#define ENC28J60_CONTROL**. Przy tej okazji od razu wykasujemy całkowicie funkcję **enc28j60clkout()**. Należy jednak pamiętać, żeby usunąć także jej deklarację w odpowiednim pliku nagłówkowym enc28j60.h. Pozostanie nam jeszcze do zmiany początek pliku, którego oryginalna wersja została przedstawiona na list 2.

Po wprowadzeniu zmian powinien on zostać skrócony do linijki: **#include <util/delay.h>**.

Teraz możemy przejść do przykładowego kodu programu serwera HTTP. Prześledzimy także zmiany, które trzeba wprowadzić w celu uruchomienia na naszych mikrokontrolerach ATmega16/32 oraz pozostałe związane z generowaniem strony WWW.

Listing 1. Definicja pinów dla poszczególnych modułów

```
#define ENC28J60_CONTROL_PORT    PORTB
#define ENC28J60_CONTROL_DDR     DDRB
#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega88P__) || defined(__AVR_
ATmega168__) || defined(__AVR_ATmega168P__) || defined(__AVR_ATmega328P__)
#define ENC28J60_CONTROL_CS PORTB2
#define ENC28J60_CONTROL_SO PORTB4
#define ENC28J60_CONTROL_SI PORTB3
#define ENC28J60_CONTROL_SCK PORTB5
#endif
#if defined(__AVR_ATmega644__)||defined(__AVR_ATmega644P__)||defined(__AVR_
ATmega32__)||defined(__AVR_ATmega16__)
#define ENC28J60_CONTROL_CS PORTB4
#define ENC28J60_CONTROL_SO PORTB6
#define ENC28J60_CONTROL_SI PORTB5
#define ENC28J60_CONTROL_SCK PORTB7
#endif
```

Tabela 1. Lista plików z funkcjami

Plik z kodem	Plik nagłówkowy	Opis
enc28j60.c	enc28j60.h	Driver dla układu ENC28J60
ip_arp_udp_tcp.c	ip_arp_udp_tcp.h	Wszystkie podstawowe funkcje stosu TCP
	ip_config.h	Konfiguracja biblioteki
	net.h	Definicje konieczne do obsługi drivera
	timeout.h	Funkcje potrzebnych opóźnień czasowych
dnslkup.c	dnslkup.h	Funkcje służące do rozpoznawania nazw przez DNS

Listing 2. Oryginalna definicja pliku nagłówkowego enc28j60.h

```
#define F_CPU 12500000UL  // 12.5 MHz
#ifndef ALIBC_OLD
#include <util/delay_basic.h>
#else
#include <avr/delay.h>
#endif
```

Na listingu 3. Przedstawiono cały kod programu serwera HTTP uruchomiony na mikrokontrolerze ATmega32. Na początku typowe jest dołączanie plików nagłówkowych zarówno tych systemowych, jak i potrzebnych bibliotek. Warto zauważyć, że oznaczyłem jako komentarz linię dołączającą plik timeout.h, który wcześniej usunęliśmy.

MAC, IP i port

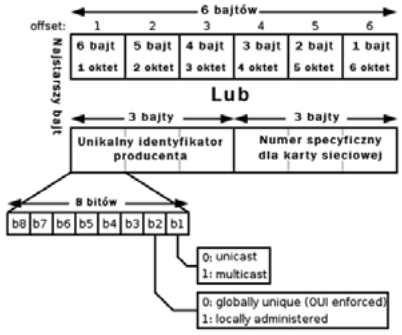
Następnie mamy do czynienia z bardzo istotną rzeczą, jaką jest przydzielenie odpowiedniego adresu MAC dla naszego urządzenia, oraz adresu IP. Trzeba pamiętać, że oba muszą być unikalne, tzn. że nie mogą przybierać takiej samej wartości jak w innym urządzeniu pracującym w twojej sieci lokalnej. Musisz, zatem sprawdzić w systemie, (np. Windows), jaki adres IP ma twój komputer. Można tego dokonać za pomocą polecenia „ipconfig /all" wydanego z linii poleceń w konsoli typu DOS. Zakładam oczywiście, że czytelnik ma dostęp do lokalnej sieci LAN, w której pracuje jakiś router, a ten wyznacza dostępną klasę adresów IP

Listing 3. Cały kod programu serwera HTTP uruchomiony na mikrokontrolerze ATmega32, pozbawiony angielskich komentarzy

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <string.h>
#include „ip_arp_udp_tcp.h"
#include „enc28j60.h"
//#include „timeout.h"
#include „util/delay.h"
#include „net.h"

// definicja własnego MAC adresu urządzenia
static uint8_t mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x29};
// definicja własnego adresu IP urządzenia
static uint8_t myip[4] = {192,168,0,110};

// port nasłuchiwania serwera www
#define MYWWWPORT 80

// bufor do obsługi transmisji TCP/HTTP
#define BUFFER_SIZE 850
static uint8_t buf[BUFFER_SIZE+1];

uint16_t http200ok(void) {
        return(fill_tcp_data_p(buf,0,PSTR(„HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nPragma: no-cache\r\n\r\n")));
}

// przygotowanie strony www poprzez jej zapis do bufora
uint16_t print_webpage(uint8_t *buf) {
        uint16_t plen;
        plen=http200ok();
        plen=fill_tcp_data_p(buf,plen,PSTR(„<pre>"));
        plen=fill_tcp_data_p(buf,plen,PSTR(„<font color='green' size='6'><b>Witaj !</b>\n</font>"));
        plen=fill_tcp_data_p(buf,plen,PSTR(„<font color='blue'><i>twój serwer www działa znakomicie</i>\n\n</font>"));
        plen=fill_tcp_data_p(buf,plen,PSTR(„<img src=http://www.atnel.pl/atnel_mini.jpg>"));
        plen=fill_tcp_data_p(buf,plen,PSTR(„<a href=http://www.atnel.pl><br>www.atnel.pl</a>"));
        plen=fill_tcp_data_p(buf,plen,PSTR(„</pre>\n"));
        return(plen);
}

// główna funkcja programu
int main(void){
        uint16_t dat_p;

//        CLKPR=(1<<CLKPCE);
//        CLKPR=0; // 8 MHZ
//        _delay_loop_1(0); // 60us

        //inicjalizacja sprzętowej karty - przypisanie adresu MAC
        enc28j60Init(mymac);
//        enc28j60clkout(2); // change clkout from 6.25MHz to 12.5MHz
//        _delay_loop_1(0); // 60us
        enc28j60PhyWrite(PHLCON,0x476);

        //inicjalizacja warstwy stosu Ethernet/IP
        init_ip_arp_udp_tcp(mymac,myip,MYWWWPORT);

        while(1){
                // odczyt pakietu, obsługa ping
    dat_p=packetloop_icmp_tcp(buf,enc28j60PacketReceive(BUFFER_SIZE, buf));

                /* dat_p będzie różny od 0 jeśli jest zapytanie  * http get */
                if(dat_p==0){
                        // brak zapytań - kontynuacja pętli
                        continue;
                }
                // tcp start obsługi portu 80
                if (strncmp("GET ",(char *)&(buf[dat_p]),4)!=0){
                        // obsługa metod POST, GET
                        dat_p=http200ok();
                        dat_p=fill_tcp_data_p(buf,dat_p,PSTR("<h1>200 OK</h1>"));
                        goto SENDTCP;
                }
                // just one web page in the "root directory" of the web server
                if (strncmp("/ ",(char *)&(buf[dat_p+4]),2)==0){
                        dat_p=print_webpage(buf);

                        goto SENDTCP;
                }else{
                        dat_p=fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 401 Unauthorized\r\nContent-Type: text/html\r\n\r\
n<h1>401 Unauthorized</h1>"));
                        goto SENDTCP;
                }
SENDTCP:
                www_server_reply(buf,dat_p); // wysłanie strony http
                // tcp koniec obsługi portu 80
        }
        return (0);
}
```

w sieci. Np. jeśli komputer będzie miał adres 192.168.0.100, niemal na pewno można urządzeniu przypisać adres IP z zakresu od: 192.168.0.1 do 192.168.0.254 pomijając adres IP, który jest przypisany komputerowi PC oraz adres tzw. bramy domyślnej. Adres tej bramy także zostanie wyświetlony w konsoli, przy czym zwykle w tej klasie domyślnie wynosi on: 192.168.0.1 ale w teorii może być on różny, w zależności od routera. Warto sprawdzić, w jakiej części klasy router przydziela adresy automatycznie, a którą jej część pozostawia dla adresów przydzielanych statycznie (przez użytkownika). Tu wszystko zależy od konfiguracji routera. Na komputerze autora router za pomocą DHCP przydziela automatycznie adresy w zakresie 192.168.0.2 do 192.168.0.99, dlatego wybrałem dowolny adres z zakresu: 192.168.0.100 do 192.168.0.254, konkretnie: 192.168.0.110. Nieco inaczej i prościej jest z dobraniem własnego adresu MAC dla naszego urządzenia. W zasadzie można wpisać 6 dowolnych przypadkowych liczb do tablicy **mymac[6]**. Można także zastosować nieco bardziej przyjazny zapis, np. w takiej postaci:

// *definicja własnego MAC adresu urządzenia*
static uint8_t mymac[6] =
{0,'M','I','R','E','K'};



Rys. 2. Zasada tworzenia adresów MAC.

Dokładnie tak samo wprowadzamy 6 cyfr, które są bezpośrednio kodami ASCII podanych w apostrofach liter. Uwagę może wzbudzić pierwszy bajt o wartości 0. Niestety autorzy z tuxgraphics.org popełnili błąd tłumacząc w kodzie sposób, jak można uzyskać własny adres MAC. W ich przypadku jest to:

static uint8_t mymac[6] =
{'T','U','X',0x10,0x00,0x24};

W rzeczywistości trzy pierwsze litery TUX zostały zaprezentowane w postaci szesnastkowej w ich przykładach:

static uint8_t mymac[6] = {0x54,0x55,0x58
,0x10,0x00,0x24};

Jednocześnie tłumaczą, jak uzyskać własny MAC adres, cytuję:

"// how did I get the mac addr? Translate the first 3 numbers into ascii is: TUX"

Co sugeruje, iż można bez cienia wątpliwości wprowadzić własny adres np.:

static uint8_t mymac[6] =
{'A','L','A',0x10,0x00,0x24};

Niestety w ten sposób przypadkowo utworzony adres MAC może powodować przedziwne zachowania się przykładowych programów w sieci lokalnej lub rozległej. Przedziwne z powodu braku znajomości prawidłowych sposobów tworzenia MAC adresów. Często właśnie przypadkowa zmiana MAC adresu w przykładowych programach z tej strony powoduje, że nie działają one zgodnie z oczekiwaniem i przez to wielu ludzi się zniechęca do dalszych prób ze stosem TCP. Początkujący nie zwraca uwagi na to, że dokonał zmiany adresu MAC, tylko na to, że kod jest identyczny jak w przykładach, a pomimo to informacje raz się pojawiają, a innym czasem nie docierają do albo z komputera. Zasada tworzenia adresów MAC została zilustrowana na rysunku 2.

Za niepowodzenia początkujących w ustalaniu adresu MAC odpowiadają dwa pierwsze najmłodsze bity z 1. oktetu. Być może autorzy tuxgraphics.org wybrali trzy pierwsze litery świadomie, szczególnie pierwszą literę „T". Kod litery T wynosi 0x54. (binarnie: 0b0101 0100), zatem dwa najmłodsze bity 1. oktetu ustawione są na 0 i taki adres MAC praktycznie w każdych

warunkach nie będzie powodował problemów z przykładami prezentowanymi przez autorów. Nie będę dalej szczegółowo opisywał, co oznacza „unicast", „multicast" czy „locally administrated", ponieważ temat nieco odbiega od naszego głównego wątku. Podpowiadam tylko, że warto, aby do takich podstawowych prób, testów obydwa bity miały wartość równą 0. Uniemożliwia to korzystanie ze wszystkich liter, gdyż w części z nich, ich kody ASCII będą miały ustawione obydwa te bity lub któryś z nich na wartość 1. Dlatego podany wyżej przykład z trzema pierwszymi literami ALA nie będzie działał w każdym przypadku zgodnie z oczekiwaniami. Reasumując, z powodów opisanych wyżej, najbezpieczniej będzie, jeśli zawsze pierwszy oktet MAC adresu będzie miał wartość 0, wtedy unikniemy przykrych niespodzianek i długiego poszukiwania błędów we własnym programie.

Przy prawidłowo dobranych numerach MAC adresu wystąpi raczej bardzo nikłe prawdopodobieństwo, że jakieś urządzenie w sieci LAN będzie miało już wybrany przez nas adres. W dalszej kolejności ustawiamy port, na którym będzie nasłuchiwał nasz serwer WWW. Typowo odbywa się to na porcie 80. Dzięki temu, we własnej przeglądarce internetowej wystarczy wpisać w miejsce nazwy strony adres IP (w moim przypadku 192.168.0.110) oraz nacisnąć klawisz ENTER. Jeśli jednak zmienimy port np. na wartość 8090, to wywołanie w przeglądarce będzie już wymagało wyspecyfikowania tegoż portu: 192.168.0.110:8090.

Definicja bufora

Kolejna kwestia dotyczy definicji bufora do obsługi komunikacji TCP. Obsługuje on zarówno dane przychodzące, jak i wychodzące. Jego wielkość zwiększyłem do 850 bajtów. Można ją jeszcze zwiększać, ale trzeba pamiętać o obserwacji zajętości pamięci RAM po kompilacji. Nie może zostać jej zbyt mało, gdyż dojdzie do problemów z działaniem stosu. W dalszej kolejności mamy definicję dwóch funkcji narzędziowych. Pierwsza z nich służy tylko do przygotowania odpowiedzi z serwera w formacie HTTP o tym, że zapytanie do serwera było poprawne, druga natomiast, print_webpage(), odpowiedzialna jest bezpośrednio za wygląd strony WWW, jaką podaje nasz serwer. Funkcja ta napełnia tylko bufor ramki TCP, natomiast samo wysyłanie odbywa się za pomocą innych funkcji w dalszej części programu. Na jej wyjściu otrzymujemy długość strony w bajtach.

Główna część programu

Następnie rozpoczyna się główna funkcja programu, main(). Na początku deklarowane są potrzebne zmienne, oraz dokonywana inicjalizacja sprzętowej części karty sieciowej, ustawianie adresów MAC oraz IP itd. Później rozpoczyna się pętla nieskończona, w której wciąż na początku dokonywany jest odczyt danych z układu ENC28J60, jeśli takie w ogóle pojawiły się na jego wejściu. Posługujemy się tutaj zmienną dat_p. Jej wartość po odczycie odpowiada ilości odczytanych bajtów. Jeśli jest równa 0 to oznacza, że nie było żadnego żądania i pętla jest kontynuowana bez wykonywania pozostałych poleceń, które są umieszczone w dalszej części programu. Trzeba pamiętać o tej konstrukcji, bo jeśli np. umieścimy jakiś własny dodatkowy kod także poniżej sprawdzania ilości odczytanych bajtów, to nie zostanie on nigdy wykonany, dopóki nie nadejdzie żądanie do serwera. Aby wybrnąć z tej sytuacji, np. gdy chcemy umieścić dla celów testowych naszą wyżej omawianą funkcję SuperDebounce(), która w zależności od stanu klawisza zapalałaby bądź gasiła diodę LED, trzeba umieścić ją w tej pętli albo przed odczytem czy sprawdzaniem zmiennej dat_p, albo nieco inaczej napisać kod całej pętli biorąc pod uwagę to, że kolejne polecenia powinny być wykonywane tylko i wyłącznie, gdy mamy jasną sytuację, że nadeszły jakieś dane.

Dalsza część programu

W dalszej części programu, jeśli nadejdzie już jakieś zapytanie z zewnątrz do naszej karty sieciowej, to po odczytaniu w buforze zostanie umieszczona ramka TCP wraz z danymi, które nas interesują. Dlatego w pierwszym warunku obsługujemy troszkę po macoszemu zapytania typu GET, natomiast w kolejnym sprawdzamy w uproszczony sposób, czy nastąpiło odwołanie do domyślnej strony w głównym folderze „/ ". Występująca tutaj spacja oznacza, że podczas wywołania naszego adresu w przeglądarce nie określono żadnej konkretnej nazwy pliku do odczytu. W przeciwnym wypadku moglibyśmy na tym właśnie poziomie rozpoznawać dostępne dla zapytań pliki, np.:

```
if (strncmp(„/index.html",(char
*)&(buf[dat_p+4]), 11)==0)
```

To spowodowałoby, że nasz serwer wyświetli stronę WWW tylko dla zapytań wpisanych w przeglądarce w ten sposób: „192.168.0.110/index.html". Można przygotować kilka różnych stron HTML w pamięci FLASH mikrokontrolera albo też na karcie pamięci SD i wysyłać je w zależności od nadchodzących żądań z zewnątrz. Ten prosty przykład nie pokazuje wprawdzie wprost możliwości sterowania mikrokontrolerem poprzez wyświetlanie strony www, ale istnieje wiele innych przykładów na stronie *tuxgraphics.org*, dzięki którym można bez problemu sterować zdalnie dowolnymi procesami.

Niestety jest jednak pewien problem, którego nie można pominąć. Otóż biblioteki stosu tcp, o których wspomniałem wyżej, są wciąż rozwijane, ale także dosyć mocno modyfikowane, co powoduje, że praktycznie w każdym pobranym przykładowym projekcie z tuxgraphics.org spotkamy się z mniej lub więcej zmodyfikowanymi funkcjami w plikach „ip_arp_udp_tcp.*". Są one czasem wprost dostosowane do konkretnego projektu, przez co niestety nie można powiedzieć, że to biblioteki w pełni uniwersalne. Nie wchodząc zatem nieco głębiej w tajniki działania stosu TCP czytelnik może mieć problemy, aby płynnie modyfikować opisane tam projekty, jeśli chodzi o wprowadzanie większych zmian w ich funkcjonowaniu. Proponuję jednak przećwiczyć i przetestować chociaż kilka projektów dostosowując je tak, jak pokazałem w tym rozdziale, do własnego mikrokontrolera. Łatwo wtedy dostrzec pewne powtarzające się zależności, z których jasno wyłoni się sposób podstawowej obsługi stosu TCP. W dużym skrócie, jego obsługa sprowadza się tak, jak w tym przykładowym omawianym tu programie prostego serwera HTTP, do ciągłego sprawdzania, czy układ enc28j60 otrzymał jakieś dane. W dalszej kolejności zwykle sprawdzamy, czy zapytanie, które dotarło, nie jest zapytaniem czysto informacyjnym związanym ustalaniem adresów, obsługi ARP, czy też obsługi zewnętrznych poleceń typu ping. W innych projektach znaleźć można sprawdzanie, czy ramka, którą odczytał układ ENC28J60, jest przeznaczona właśnie dla nas, i dopiero po tym następują procedury parsowania (sprawdzania) danych przesłanych w ramce, które mogą być przydatne do sterowania procesami. Obsługa stosu już w kodzie programu, mając do dyspozycji chociaż podstawowe funkcje, nie jest wcale taka trudna. Dodam że chociaż na schematach przykładowych urządzeń z tuxgraphics.org prawie wszędzie podłączony jest sygnał INT z karty sieciowej do wejścia INTx mikrokontrolera, to jednak w kodzie jest ono nie używane. A szkoda. Można sobie wprowadzić taką obsługę, wystarczy zainicjalizować to przerwanie, w naszym zestawie będzie to wejście INT2 i napisać najprostszą obsługę przerwania w postaci ustawiania zwykłej flagi, która będzie następnie sprawdzana i zerowana w pętli głównej. Oznaczać ona będzie zdarzenie odbioru danych z karty sieciowej. Można dzięki temu nieco uprościć oprogramowanie w pętli głównej programu, zamiast wciąż badać, czy bufor jest pusty. Wprawdzie w tak prostym przykładzie nie robi to wielkiej różnicy, jednak daje pewne możliwości dla przyszłej wygodniejszej rozbudowy całego programu.

Mirosław Kardaś
Atnel