Ethernet w jednym module, część 4

Taiko: Ethernet w Basicu

Implementacja sprzętowej bramki SMS

Przechodzimy do praktycznej części naszej prezentacji: przedstawimy opis konfiguracji modułu EM202, za pomocą której uzyskamy bramkę SMS z interfejsem Ethernet. Na płycie CD–EP5/2007B (oraz na stronie internetowej taiko.ep.com.pl) publikujemy kompletny projekt, który – dzięki Taiko – został napisany w Basicu.

Źródła projektu opublikujemy na CD–EP5/2007B. W folderze *hw_sms_www* znajdują się wszystkie pliki źródłowe łącznie z plikami *HTML*. Uruchamiamy więc Tibbo IDE (także dostępne na CD–EP5/2007B) i otwieramy projekt *Ep.tpr*. Kod został oparty na szablonie *DHCP +setup Project* dostępnym w TIDE. Zawiera on implementację klienta DHCP oraz część inicjalizującą sockety i UART modułu EM202.

Zaczniemy od inicjalizacji. Przechodzimy zatem do pliku *init.tbs*, w którym znajduje się handler zdarzenia generowanego tuż po zerowaniu modułu – *on_sys_init*. Duża część kodu zawartego w tym pliku pochodzi ze wspomnianego szablonu. Dla nas najważniejsze jest zdecydować, co należy zrobić z modułem po starcie.

W najgorszym wypadku w sieci, w której będzie pracował moduł, nie będzie serwera DHCP. Musimy zatem sami zatroszczyć się o nadanie mu odpowiedniej konfiguracji IP. Odnajdujemy w źródłach fragment pokazany na **list. 3**.

Interesuje nas jedynie zaznaczony na szaro fragment instrukcji *if* (w dalszej części wyjaśnię całość zapisu). Jeśli nie udało się uzyskać konfiguracji IP z serwera DHCP, ustawiamy wartość własności *net.ip* oraz *net.netmask* na stałe w kodzie. Moduł po nieudanej próbie komunikacji z serwerem użyje tych domyślnych ustawień. Zastosowany szablon alokuje też miejsce w pamięci na bufory dla gniazd HTTP. Ilość socketów determinuje liczbę klientów obsługiwanych naraz przez serwer, naturalnym jest więc, że liczba ta powinna być jak największa (jednak nie większa niż 15). Z drugiej strony ogranicza nas własność *sys.freebuffpages*. Musimy też zaalokować bufory do obsługi UART–u, który przecież będzie komunikował się z modemem.

List. 3. Akcja podejmowana w przypadku braku komunikacji z serwerem DHCP

```
'could not complete DHCP- you decide what to do here
'For example, can set fixed IP OR DO SOMETHING ELSE
     if (stor.get(61,1) = "T") then
        net.ip=stor.get(0,20)
        net.netmask=stor.get(20,20)
        net.gatewayip=stor.get(40,20)
     else
        net.ip="192.168.3.92"
        net.netmask="255.255.255.0"
  end if
```

W używanym szablonie zastosowano następujący podział dostępnych stron pamięci: 1/3 przypada na obsługę socketu 0 przeznaczonego np. do diagnostyki – nie obsługuje on połączeń http, 1/3 przypada na obsługę UART–u i 1/3 zostaje przydzielona pozostałym socketom. Podział odbywa się dynamicznie, a absolutny rozmiar fragmentu pamięci (1/3) zostaje przekalkulowany w ten sposób, aby obsłużyć jak największą liczbę socketów HTTP. W szablonie przydzielono po jednej stronie pamięci do każdego z buforów: odbiorczego, nadawczego i zmiennych, jednak dobrze jest ten rozmiar zwiększyć do 3. Wywołując metodę *sys.buffalloc* dokonujemy właściwej alokacji pamięci.

Kolejnym krokiem jest konfiguracja portu szeregowego, co pokazano na **list. 4**.

Komentarza wymaga jedynie podświetlona na szaro linijka: wła-

List. 4. Konfiguracja układu UART

```
'Setup the serial port
  ser.num=0
  ser.baudrate=ser.div9600
  ser.bits=PL_SER_BB_8
     ser.parity=PL_SER_PR_NONE
     ser.flowcontrol=DISABLED
     ser.interface=PL_SER_SI_FULLDUPLEX
     ser.enabled=YES
```

List. 5. Inicjalizacja modemu Maestro 20

```
'Wismo init
  echo_off()
  serial_command = COMM_ECHO
  conf_state = CONF_IDLE
```

List. 6. Definicja funkcji echo_off()

```
public function echo_off() as
boolean
  ser.setdata("ate"+chr(13)+chr(10))
  ser.send
end function
```

List. 7. Definicja typu *comm_state*

```
enum comm_state
  COMM_IDLE,
  COMM_ECHO,
  COMM_PIN,
  COMM_CHECK_PIN,
  COMM_CHANGE_PIN,
  COMM_SEND_SMS
End enum
```

sność *ser.div9600* zwraca liczbę całkowitą, jaką należy przypisać innej własności – *ser.baudrate*, aby port szeregowy pracował z prędkością 9600 b/s. Jest to zabieg twórców TiOS na unifikację kodu programu, który docelowo miałby pracować na różnych platformach sprzętowych (czyli z innym zegarem taktującym układ UART). W tym momencie platforma została zainicjalizowana.

Teraz czas na układ peryferyjny, a więc moduł Maestro 20 z układem Wismo (**list. 5**).

Przechodzimy do pliku *wismo.tbs* i odnajdujemy definicję funkcji *echo_off()* – **list. 6**.

Metoda *ser.setdata* przyjmuje jako argument łańcuch znaków, który zostaje następnie przepisany do bufora nadawczego UART–u. W całym zapisie użyłem dwukrotnie funkcji systemowej *chr()*, która konwertuje podany w jej argumencie kod znaku ASCII na właściwy znak ASCII. Kody 13 i 10 odpowiadają znakom powrotu karetki i nowej linii, co z kolei jest interpretowane przez modem Maestro jako zatwierdzenie komendy (*ate* – wyłączenie echa). Konkatenacja łańcuchów znaków w Tibbo BASIC odbywa się przy użyciu operatora „+". Kiedy dane są gotowe do wysłania, wywołujemy metodę *ser.send*. Pomimo tego, że wyłączyliśmy echo komend, przestanie ono być wysyłane dopiero przy następnej komendzie, zatem spodziewamy się odpowiedzi: 'CR'+'LF'+ATE OK+'CR'+'LF'. Wskutek tego powinno zostać wygenerowane zdarzenie: *on_ser_data_arrival*. I tu pojawia się mały problem: komunikacja z modemem będzie odbywać się ciągle z użyciem różnych komend, musimy zatem skonstruować uniwersalny odbiornik, który będzie analizował odpowiedzi i podejmował dalsze akcje. Zrealizowane jest to przy pomocy publicznej zmiennej o nazwie *serial_command*. Zmienna ta jest typu wyliczeniowego, który przyjmuje wartości jak na **list. 7**.

Widać, że po wyłączeniu echa (a więc po wysłaniu komendy do modemu) *serial_command* przyjmuje wartość COMM_ECHO. Przejdźmy teraz do pliku *serial.tbs* i odnajdźmy fragment z list. 6. W handlerze zdarzenia odbioru widoczna jest struktura *'select case'* przełączająca się zgodnie z wartością zmiennej *serial_command*.

List. 8. Definicja typu *comm_state*

```
sub on_ser_data_arrival
  serial_buf = ser.getdata(30)
  select case serial_command
     case COMM_IDLE:
        serial_command = COMM_IDLE
     case COMM_ECHO:
        if serial_buf = (chr(13) + chr (10) + "ATE OK" +
                         chr(13) + chr (10)) then
                         'echo was on
        else
           if serial_buf = (chr(13) + chr (10) + "OK" +
                            chr(13) + chr (10)) then
                            'echo was off
           end if
        end if
        ask_for_pin()
        serial_command = COMM_PIN
```

W przypadku COMM_ECHO mamy dwie możliwości: moduł GSM został włączony razem z EM202 i ma włączone echo (ustawienie domyślne), bądź z jakichś przyczyn modem miał już wyłączone echo, więc spodziewamy się tylko odpowiedzi OK (oraz dwukrotnie znaków CR i LF). W każdym przypadku kolejnym krokiem inicjalizacji będzie zapytanie o potrzebę wprowadzenia kodu PIN.

Zadanie to realizuje funkcja *ask_for_pin()* wysyłająca do modułu komendę: *at+cpin?*. Zmienna sterująca odbiornikiem UART–u przyjmuje wartość COMM_PIN. Jeśli karta SIM modułu nie potrzebuje pinu (odpowiedź: '+CPIN: READY'), bądź został on już wpisany, wówczas *serial_command* będzie równa COMM_IDLE. W przeciwnym przypadku wywołana zostanie funkcja *enter_pin()* wprowadzająca właściwy kod karty. Na tym etapie moduł jest gotowy do pracy.

Interfejs WWW

Nadeszła pora na stworzenie interfejsu WWW naszej sprzętowej bramki SMS. Do projektu dołączonych jest 5 plików HTML, z czego 3 są kluczowe. Najważniejszy z nich to plik *index.html* ładujący się w przeglądarce jako strona główna (**rys. 11**). Jest ona skonstruowana jako formularz (a w zasadzie dwa) posługujący się metodą GET http do przekazania parametrów. Znaczniki:

<form action="send.html" method="get">

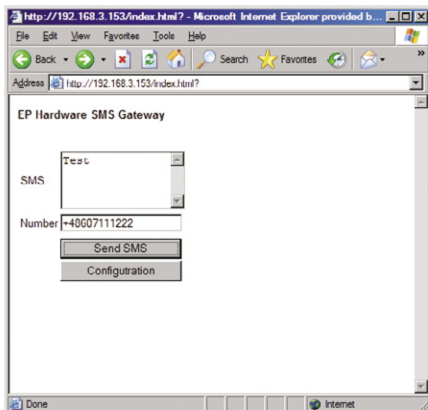
<form action="conf.html" method="get">

determinują podstrony, do których zostaną przekazane dane z formularzy w przypadku użycia przycisku *Send SMS* bądź *Configuration*. Prześledźmy teraz, co dzieje się, gdy chcemy wysłać SMS–a.

Po wypełnieniu pól treści SMS i numeru klikamy przycisk *Send SMS*, co powoduje wysłanie przez przeglądarkę do serwera żądania o stronę *send.html* w postaci:

http://192.168.3.153/send.html?sms=Test&number=%2B48607111222

Nie byłoby w tym nic niezwykłego, gdyby nie kod źródłowy strony *send.html* (jego część pokazano na **list. 9**). Pole zaznaczone na szaro, pomiędzy znacznikami <?></?> to

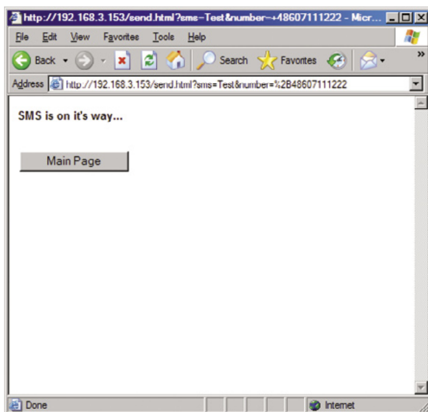


Rys. 11. Główna strona bramki

nic innego, jak skrypt Tibbo BASIC wykonywany po stronie serwera. Po deklaracjach potrzebnych zmiennych publicznych następuje przepisanie do *s* zawartości bufora zmiennych http, czyli ciąg: html?sms=Test&number=%2B48607111222 (własność *sock.httprqstring*) oraz przypisanie: *command = COMM_SEND_SMS*. I to wszystko...

Nie oznacza to jednak jeszcze, że SMS został wysłany (pomimo tego, iż tak głosi komunikat z **rys. 12**). Przejdźmy teraz do pliku *main.tbs*. Widzimy tam handler zdarzenia *on_sys_timer* generowanego co 0,5 s. Wewnątrz widnieje:

```
if serial_command = COMM_SEND_SMS then
    serial_command = COMM_IDLE
    send_sms(s)
end if
```



Rys. 12. Informacja o wysłaniu SMS–a

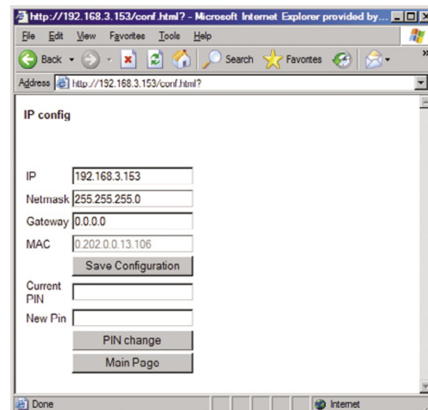
A więc skrypt w *send.html* wystawił flagę, która jest periodycznie (co pół sekundy) sprawdzana w głównym module programu. Jeśli flaga wystąpiła, to zostaje wywołana właściwa funkcja wysyłająca SMS–a, a przyjmująca jako parametr publiczną zmienną *s*. Definicja tej funkcji znajduje się w *wismo.tbs*. Ekstrahuje ona ze zmiennej HTML pola dotyczącego treści wiadomości oraz numeru telefonu (przy pomocy *extract_content()* i *extract_number()*), a ponieważ elementy te są zakodowane (zgodnie z HTML), to funkcja ta wywołuje kolejną funkcję: *decode_html()*. Kiedy już treść i numer telefonu są w jawnej postaci, następuje konstrukcja komendy modemu GSM służącej do nadania wiadomości, a następnie wysłanie całości przy pomocy metody *ser.send*.

Bramka została też wyposażona w ekran konfiguracyjny. Po kliknięciu przycisku *Conifiguration* powinniśmy ujrzeć stronę pokazaną na **rys. 13**. Mamy zatem do dyspozycji konfigurację protokołu IP, jak również opcję zmiany kodu PIN karty SIM modułu GSM. Akcje podejmowane po użyciu przycisków *Save Configuration* i *PIN change* przebiegają analogicznie jak przy wysyłaniu SMS–a, a więc najpierw następuje przepisanie zawartości formularza do publicznej zmiennej i następnie ustawienie odpowiedniej flagi (tylko tyle dzieje się w skrypcie w kodzie HTML). Moduł główny, tak jak poprzednio zajmie się całą resztą, nie obciążając tym zadaniem interfejsu użytkownika. Dokładniejszego omówienia wymaga nie zmiana konfiguracji IP, ale jej zachowanie

List. 9. Fragment kodu strony *send.html*

```
<?
include "global.tbh"
?>

<!DOCTYPE HTML public "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
    <basefont color="black" face="arial" size="2">
    <?
    declare s as string
    declare serial_command as comm_state
    s = sock.httprqstring
    serial_command  = COMM_SEND_SMS
    ?>
```



Rys. 13. Okno konfiguracji bramki

w nieulotnej pamięci (w przypadku EM202 jest to EEPROM).

Przejdźmy do pliku *ep202ip.tbs*. Znajduje się w nim definicja funkcji *storeip()*, która podobnie jak *send_sms()* wyłuskuje ze zmiennej HTML potrzebne adresy i przypisuje je kolejno własnościom obiektu *net*, po czym zachowuje je w pamięci EEPROM za pomocą metody *stor.set()*. Metoda ta przyjmuje jako argumenty *string* do zachowania, jak również adres początkowy. Na końcu użytego obszaru funkcja *storeip()* wpisuje wartość ‘T’. Właśnie ta komórka pamięci była sprawdzana na samym początku (list. 3). Oznacza ona bowiem, że adres IP został zmieniony przez użytkownika i moduł ma przypisać sobie właśnie adres IP zapisany w pamięci EEPROM.

Podsumowanie

Pomysł twórców TAIKO na zaimplementowanie w systemie wbudowanym metody programowania dostępnej dotychczas tylko w komputerach klasy PC oceniam jako trafny, a samą implementację za udaną. Składnia języka, funkcje systemowe i sam przebieg wykonania programu są niezwykle intuicyjne i sprawiają, że program tak naprawdę „sam się pisze”. Tibbo zapowiada też wprowadzanie nowych platform sprzętowych, co w połączeniu z ich programowalnością może stanowić silne narzędzie w obszarze interfejsów ethernetowych.

Marcin Chruściel, EP
marcin.chrusciel@ep.com.pl

Dodatkowe informacje

Soyter Sp. z o.o., tel. 022 752 82 55
www.soyter.pl, handlowy@soyter.pl