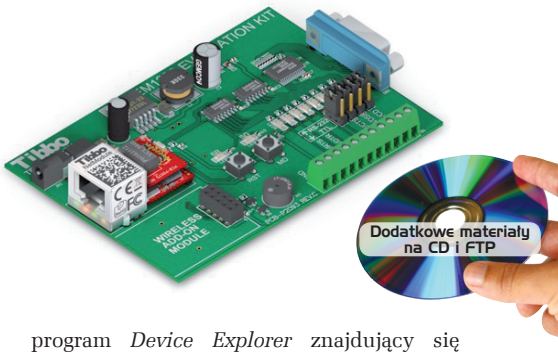
Programowanie modułu Tibbo em1206



W listopadowym numerze „Elektroniki Praktycznej" był szczegółowo opisany „inteligentny" interfejs firmy Tibbo – em1206. Dziś zaczniemy krok po kroku przygodę z programowaniem wyżej wymienionego modułu. Dowiemy się, jakie narzędzia dostarcza producent oraz napiszemy pierwszy, przykładowy program sterujący diodami LED. Uruchomimy też serwer WWW. Nic nie stoi na przeszkodzie, aby diody LED z przykładu aplikacji zastąpić np. przekaźnikami i wykonać funkcjonalne, komercyjne urządzenie.

Do celów testowych posłużymy się płytą ewaluacyjną EM1206-EV. Jest ona wyposażona między innymi w:

- moduł Tibbo EM1206 – 512K-00 z modułem złącza RJ203,
- złącze SPI do podłączenia modułu Wi-Fi – GA1000,
- kondensator podtrzymujący zasilanie o pojemności 4 F,
- buzzer,
- 10 diod świecących LED (2 kontrolne+8 dowolnie konfigurowalnych),
- interfejs RS232 ze złączem DB9,
- złącza ARK z doprowadzonymi liniami portu szeregowego w standardzie TTL lub RS232, w zależności od ustawień jumperów.

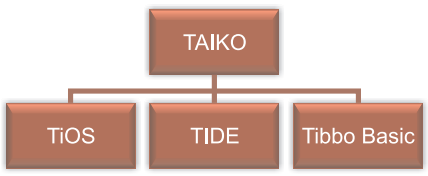
Do zasilenia płyty ewaluacyjnej należy zastosować zasilacz stabilizowany o napięciu wyjściowym 9...12 V i prądzie powyżej 500 mA. Jeżeli będziemy chcieli zasilić urządzenie zewnętrzne, korzystając z dostępnego wyprowadzenia, to należy użyć odpowiednio mocniejszego zasilacza.

TAIKO

Wraz z produktami Tibbo jest dostarczane następujące darmowe oprogramowanie:

- TiOS – system operacyjny Tibbo w wersji *„EM1206 Platform Firmware"* (*http://tibbo.com/downloads/basic/firmware.html*),
- TIDE – zintegrowane środowisko programistyczne (*http://tibbo.com/downloads/basic/software.html*),
- DST – zarządzanie wirtualnymi portami szeregowymi, np.: COM1 (*http://tibbo.com/downloads/soi/tdst.html*).

Wszystkie wymienione wyżej programy są dostępne w ramach pakietu noszącego nazwę handlową TAIKO. Zestaw ten umożliwia tworzenie i rozwijanie aplikacji oraz ich implementację w praktycznych zastosowaniach.



TiOS. Zintegrowany system operacyjny wykonany przez Tibbo Technologies. Odpowiada on za dwa główne procesy:

- *Master Process* odpowiedzialny za komunikację oraz obsługę zdarzeń,
- *Virtual Machine,* który jest kontrolowany przez nadrzędny *Master Process*. VM wykonuje polecenia programu użytkownika.

TIDE. TIDE to zintegrowane środowisko umożliwiające zarządzanie projektami oraz oczywiście – pisanie, edycję, symulację oraz wyszukiwanie błędów w programie. Moduł edytora automatycznie dokańcza komendy systemowe. Jak każde nowoczesne środowisko programistyczne, TIDE ma wbudowany kompilator, który podczas kompilacji wyświetla komunikaty o błędach składni programu.

TibboBasic. Język programowania opracowany przez Tibbo, dedykowany do programowania modułów. Tibbo Basic jest podobny do innych języków BASIC (np. VSBasic, QuickBasic, itp.), ale producent wyposażył go w dodatkowe funkcje.

Tak w dużym skrócie można opisać środowisko pracy z modułami Tibbo. W dalszej kolejności zajmiemy się napisaniem przykładu programu w języku Tibbo Basic oraz jego uruchomieniem.

Sterowanie diodami LED

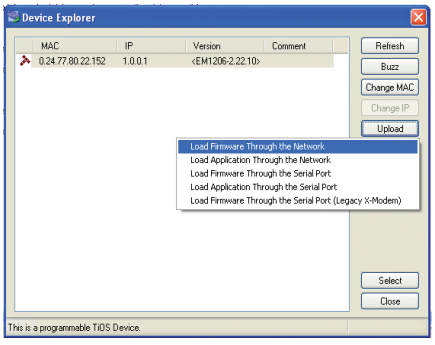
Aby zacząć, należy pobrać wyżej wymienione oprogramowanie ze strony producenta *http://www.tibbo.com*. Należy pamiętać, aby zawsze korzystać z najnowszej, dostępnej wersji.

Po instalacji *Tibbo IDE* oraz *Tibbo DST* możemy rozpocząć tworzenie aplikacji. Zaczniemy od odnalezienia modułu podłączonego do sieci oraz wgrania do niego systemu operacyjnego TiOS. W tym celu otwieramy program *Device Explorer* znajdujący się w katalogu *Tibbo\Tibbo IDE „Menu Start"*.

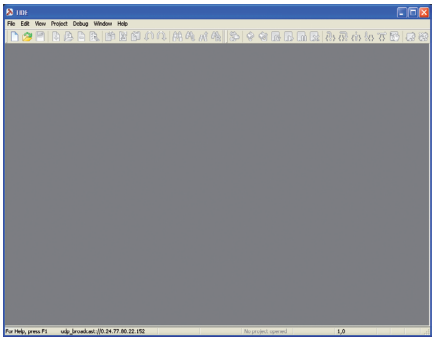
Przy pierwszym uruchomieniu należy pamiętać o ustaleniu reguły firewalla.

Z listy wykrytych modułów w sieci (**rys. 1**) zaznaczamy interesujący nas em1206. Jeżeli nie jesteśmy pewni, możemy zaznaczyć każdy z modułów i przycisnąć klawisz *Buzz*, co spowoduje szybkie miganie diody na wybranym module. Gdy jesteśmy pewni wyboru, możemy przystąpić do aktualizacji oprogramowania. W chwili pisania artykułu najnowsza wersja była oznaczona jako *tios-em1206-2_20_05.bin*.

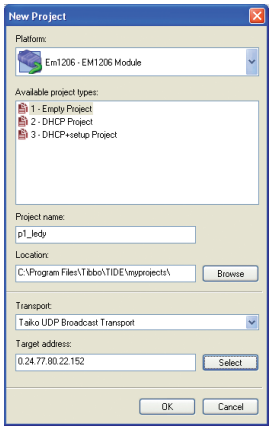
Z menu kontekstowego klawisza *Upload* wybieramy *Load Firmware Through the Network*, następnie wybieramy wcześniej pobrany plik binarny i przyciskamy klawisz *Otwórz*. Aktualizację oprogramowania modułu można wykonać również przy użyciu portu szeregowego (COMx) oraz kabla typu null-modem. O poprawnie zakończonej aktualizacji zostaniemy poinformowani odpowiednim komunikatem.



Rys. 1.

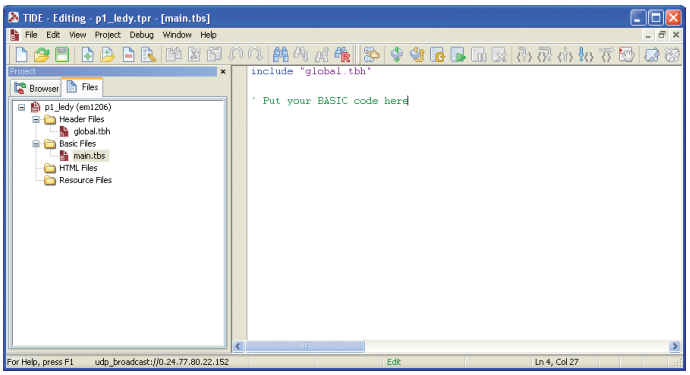


Rys. 2.

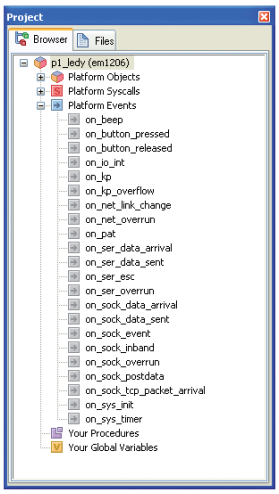


Rys. 3.

– Platform: <*Em1206 – EM1206 Module*> – definiuje platformę, na której zamierzamy uruchomić nasz program.
– Available Project types: <*Empty Project*> – szablon projektu zawierający plik główny nagłówkowa global.tbh (Tibbo Basic Header) oraz plik źródłowy main.tbs (Tibbo Basic Source). Pozostałe szablony zawierają odpowiednio dodatkowe funkcje: <*DHCP Project*> global.tbh + main.tbs + dhcp.tbs – implementacja klienta DHCP, <*DHCP + setup Project*> zawiera dodatkowo kod inicjalizacji socketów TCP i portów szeregowych w treści main.tbs.



Rys. 4.



Rys. 5.

– Project name**:** <*nazwa naszego nowego projektu*>.
– Location: <*lokalizacja folderu z naszymi projektami*>.
– Transport: <*Taiko UDP Broadcast Transport*>.

Aby uzupełnić pole *Target*, należy przycisnąć klawisz *Select*. Pojawi się znane nam okno *Device Explorer*. Tym razem jednak zaznaczamy tylko interesujący nas moduł i przyciskamy klawisz *Select*.

Na tym etapie zakończyliśmy konfigurację sprzętową projektu. Sądzę, że programiści mikrokontrolerów docenią prostotę opisanych wyżej czynności.

Na koniec potwierdzamy wprowadzone dane, przyciskając klawisz *OK*. Teraz już możemy zacząć programowanie. Na **rys. 4** pokazano główne okno programu gotowego do pracy. Po lewej stronie, w *Inspektorze Projektu*, znajduje się zakładka *Files*. Tutaj widzimy wszystkie pliki, które możemy stosować w aplikacji. Program rozpoznaje pliki własne (*.tbc; *.tbs; *.tbh) oraz pliki obce (*.txt; *.html; *.jpeg; *.jpg; *.gif; *.png; itd.).

Dalszą część pracy wykonamy już w środowisku programistycznym *Tibbo IDE*. Uruchomione środowisko przywita nas pustą przestrzenią MDI (**rys. 2)**. Zaczniemy od założenia projektu. W tym celu należy użyć Menu programu *File –> New Project*. Spośród wielu opcji należy wybrać następujące:

Rys. 6.

Klikając na drugą zakładkę, zobaczymy wszystkie możliwe zdarzenia, funkcje, zmienne globalne, z których możemy korzystać w czasie tworzenia programu. Zarówno obiekty, zdarzenia, jak i wywołania są zależne od wybranej platformy sprzętowej (**rys. 5**).

Jeżeli zatrzymamy kursor przez chwilę nad wybranym zdarzeniem, to pojawi się podpowiedź w postaci chmurki z krótkim opisem (**rys. 6**).

Dwukrotne przyciśnięcie wybranego zdarzenia powoduje stworzenie szablonu w treści otwartego źródła, w miejscu gdzie aktualnie znajduje się kursor.

W **tab. 1** znajduje się zestawienie wszystkich możliwych zdarzeń dla systemu em1206.

Po załączeniu zasilania lub zerowaniu moduły Tibbo wymagają konfiguracji. W tym celu skorzystamy ze zdarzenia *on_sys_init*. Klikamy dwukrotnie na etykietę *event*, znajdującą się w menu zakładki Browers. Spowoduje to pojawienie się szablonu w treści otwartego pliku źródłowego (w naszym przypadku w *main.tbs*). Nasze urządzenie będzie pracować w sieci Ethernet jako serwer WWW, więc musimy wpierw skonfigurować protokół IP. W aplikacji moduł będzie miał przypisany na stałe adres IP. Wpisujemy go do zmiennej *net.ip* (wartość domyślna to 127.0.0.1). W treści funkcji inicjalizującej wpisujemy np. „*net.ip = 192.168.0.95*”. Teraz nasz moduł po skompilowaniu będzie widoczny w sieci pod tym adresem. Następnie należy podać liczbę *socketów*, odpowiadającą liczbie jednoczesnych połączeń z różnymi klientami. W naszym przypadku wystarczą dwa. Każdy z socketów należy jeszcze skonfigurować, tzn. określić, jakie typy połączeń może obsłużyć, wielkości buforów oraz skąd może odebrać połączenie. Służą temu kolejne odwołania systemowe, skierowane do obiektu *sock*:

– *syscall sock.rxbuffrq(numpages as byte) as byte*, przydzielenie miejsca w pamięci dla bufora odbioru w stronach, gdzie jedna strona odpowiada 256 bajtom. Należy jednak pamiętać, że realnie mamy do dyspozycji mniejszy o 16 bajtów rozmiar zalokowanej pamięci, zgodnie ze wzorem *numpages*256-16*. Wynika to z własności systemu TiOS, który potrzebuje 16 bajtów pamięci do kontrolowanie bufora.
– *syscall sock.txbufferq(numpages as byte) as byte*, analogicznie do bufora odbiorczego deklaruje wielkość bufora nadawania.
– *syscall sock.varbufferq(numpages as byte) as byte*, deklaruje przydzielenie miejsca w pamięci dla bufora obsługi http. Wielkość ustalamy, zakładając maksymalny string przesyłany w metodzie Get.

Tab. 1. Wykaz zdarzeń systemu em1206

Nazwa zdarzenia	Argumenty	Opis
on_beep	brak	
on_button_pressed	brak	Zdarzenie wykonywane przy zmianie stanu linii Mode (zbocze opadające), służącej do wprowadzenia trybu aktualizacji firmwaru poprzez port szeregowy. Na płycie ewaluacyjnej przycisk ten można dowolnie oprogramować
on_button_released	brak	Jak wyżej, działa przy zboczu narastającym
on_io_int	linestate as byte	Generowany w momencie wykrycia zmiany stanu linii określonych, jako linie przerwań
on_kp	key_event as pl_kp_event_codes, key_code as byte	Zdarzenie wykonywane, gdy którykolwiek z klawiszy klawiatury zostanie przyciśnięty
on_kp_overflow	brak	Zdarzenie wykonywane w przypadku przepełnienia bufora klawiatury, informuje nas o możliwości zgubienia kolejnych znaków
on_net_link_change	brak	Zdarzenie wykonywane przy zmianie stanu linku warstwy fizycznej połączenia sieciowego
on_net_overrun	brak	Zdarzenie wykonywane w przypadku przepełnienia buforu odbiorczego interfejsu sieciowego – układu NIC (DM9000)
on_pat	brak	Zdarzenie wykonywane po zakończeniu wykonywania metody play na obiekcie pat, po sekwencyjnym wysterowaniu diod LED SR i SG
on_ser_data_arrival	brak	Zdarzenie wykonywane, gdy w buforze odbiorczym portu szeregowego (UART) modułu em1206 znajduje się dana
on_ser_data_sent	brak	Zdarzenie wykonywane analogicznie do powyższego, z tą różnicą, że badany jest bufor nadawczy portu szeregowego
on_ser_esc	brak	Zdarzenie wykonywane po wykrycie znaku zdefiniowanego poprzez użycie ser.escchar(), w buforze odbiorczym
on_ser_overrun	brak	Zdarzenie wykonywane w przypadku przepełnienia bufora odbiorczego portu szeregowego modułu em1206
on_sock_data_arrival	brak	Zdarzenie wykonywane w momencie pojawienia się danych w buforze odbiorczym zadeklarowanego socketu. W przypadku socketu obsługującego protokół TCP, gdy program nie zdąży odczytać całego strumienia danych, generowane jest kolejne zdarzenie, a dane nie są tracone. W przypadku UDP nieodczytane dane są tracone
on_sock_data_sent	brak	Zdarzenie wykonywane analogicznie do powyższego, z tą różnicą, że badany jest bufor nadawczy połączeń sieciowych
on_sock_event	newstate as pl_sock_state, newstatesimple as pl_sock_state_simple	Zdarzenie wykonywane po zmianie stanu socketu
on_sock_inband	brak	Zdarzenie występuje, jeżeli włączona jest obsługa komend inband i zostanie rozpoznana komenda
on_sock_overrun	brak	Zdarzenie wykonywane w chwili przepełnienia buforu odbiorczego. Wykonywane przy komunikacji po UDP
on_sock_postdata	brak	Zdarzenie generowane w momencie gdy ostatni bajt jest przesłany do VAR bufora, dla połączeń HTTP
on_sock_tcp_packet_arrival	len as word	Zdarzenie informujące nasz program, że ramka o określonej długości została odebrana, dotyczy protokołu TCP
on_sys_init	brak	Zdarzenie wykonywane tuż po zerowaniu systemu (restart lub załączenie do sieci)
on_sys_timer	brak	Zdarzenie generowane co 500ms (default) lub co minimum 10ms przy zmianie wartości parametru sys.timercount

– *syscall sys.buffalloc()* – alokuje wcześniej zadeklarowane przestrzenie w pamięci

Po ustaleniu buforów należy skonfigurować własności działania komunikacji:

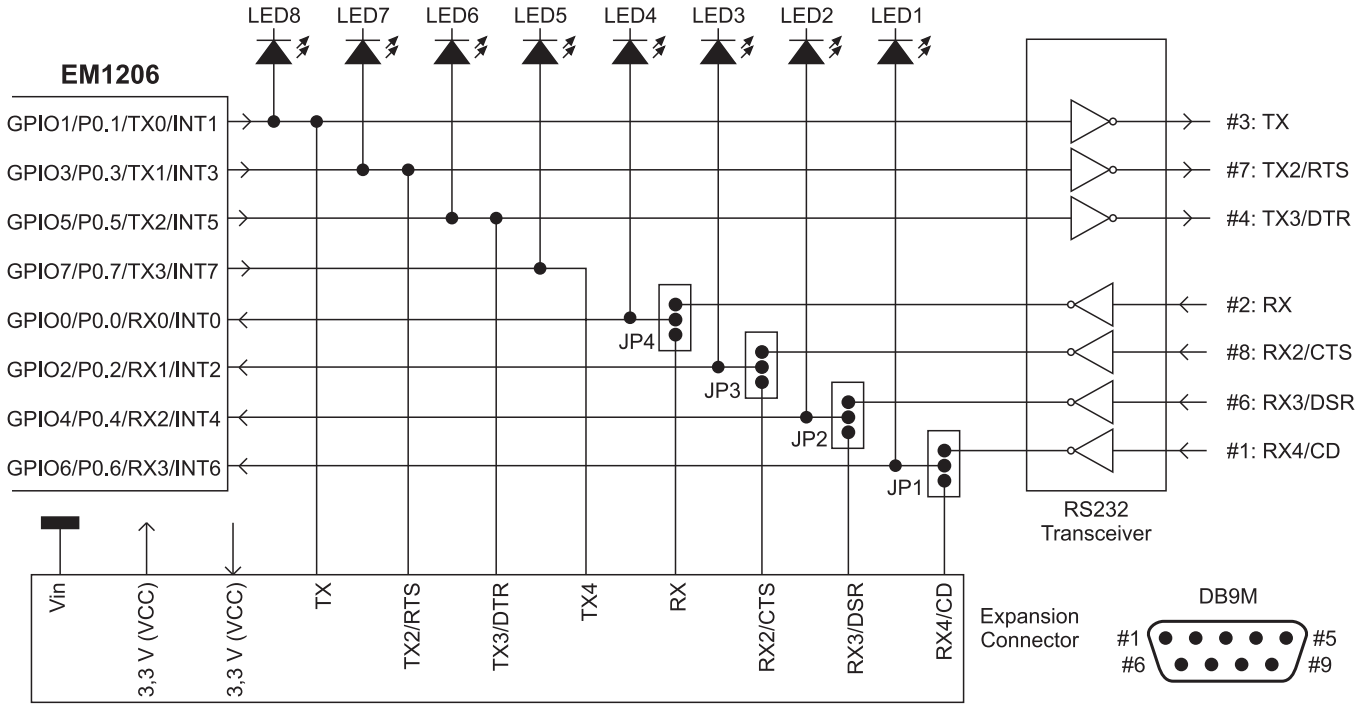
– *property sock.protocol as pl_sock_protocol* – protokół transportowy socketu. Ustawiamy TCP.

– *property sock.inconmode as pl_sock_inconmode* – własność gniazda odpowiadająca za akceptację połączeń. Mamy do wyboru następujące wartości:

- *sock.inconmode = PL_SOCK_INCONMODE_NONE*, socket nie będzie akceptował połączeń
- *sock.inconmode = PL_SOCK_INCONMODE_SPECIFIC_IPPORT*, socket będzie akceptować połączenia z określonego adresu IP (zmienna zawiera-

List. 1. Kod inicjalizacji modułu em1206, main.tbs:

```
'----------------------------
'Inicjalizacja systemu
'----------------------------
sub on_sys_init()
                                                    'zdarzenie wykonywane po zerowaniu (restart, włączenie
                                                    'zasilania)
   net.ip = „192.168.0.95”                          'stały adres IP

   dim i as byte                                    'zmienna i typu byte

   for i = 0 to 2                                   'pętla wykonywana 2-krotnie
     sock.num = i

     sock.rxbuffrq(4)                               'bufor odbioru
     sock.txbuffrq(4)                               'bufor nadawczy
     sock.varbuffrq(4) ,bufor http
     sock.protocol = PL_SOCK_PROTOCOL_TCP           'protokół TCP
     sock.inconmode = PL_SOCK_INCONMODE_ANY_IP_ANY_PORT  'akceptuje wszystkie połączenia
     sock.allowedinterfaces = „NET”                 'akceptuje połączenia LAN
     sock.httpportlist = “8080, 80”                 'numery portów dla połączenia http
   next i

   sys.buffalloc                                    'alokacja pamięci

end sub
```



Rys. 7.

List. 2. Licznik binarny

```
'------------------------
'Licznik binarny
'------------------------
sub on_button_pressed()
        i = i+1
        if i>8 then i=0

        io.portnum=PL_IO_PORT_NUM_0        'włączenie obsługi port
        io.portenabled = &hFF              'wszystkie linie jako wyjścia
        io.portstate = i
end sub
```

jąca adres *sock.targetip*) oraz PORTU (zmienna zawierająca nr portu *sock.targetport*)

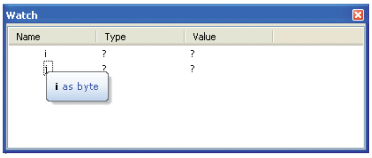
- *sock.inconmode = PL_SOCK_INCONMODE_SPECIFIC_IP_ANY_PORT*, socket będzie akceptować połączenia z dowolnego IP, natomiast port musi zostać określony w zmiennej *sock.targetport,*
- *sock.inconmode = PL_SOCK_INCONMODE_ANY_IP_ANY_PORT*, socket będzie akceptować połączenia z określonego IP w zmiennej *sock.targetip*, natomiast port może być dowolny,
- *property sock.allowedinterfaces as string* – własność definiująca, który z interfejsów będzie akceptowany („NET” – LAN, „WLN” – WLAN), mogą być wpisane po przecinku,
- *property sock.httpportlist as string* – określa numer portu, pod którym będzie można nawiązać połączenie http.

Na **list. 1** pokazano fragment programu zawierający inicjalizację modułu.

Po procesie inicjalizacji możemy napisać krótką aplikację 8-bitowego licznika binarnego, liczącego wciśnięcia klawisza MD. Wynik będzie sygnalizowany z użyciem diod LED. Należy zwrócić uwagę na numerację portu i LED-ów, ponieważ nie jest zachowana kolejność.

W celu obsługi zdarzenia przyciśnięcia klawisza MD (Mode) należy kliknąć dwukrotnie na event znajdujący się w zakładce *Browers* okna *Inspektora Projektu*. Diody LED są podłączone do linii portu szeregowego (**rys. 7**), który może też pełnić rolę zwykłego, 8-bitowego portu I/O. Aby uaktywnić 8-bitowy styl zapisu, należy posłużyć się własnościami obiektu I/O:

- *property io.portnum as pl_io_port_num* – wybranie aktywnego portu, działanie podobne jak własności io.num,
- *io.portnum=PL_IO_PORT_NUM_0* – powoduje użycie linii portu 0 na płycie ewaluacyjnej połączonej z portem szeregowym i diodami LED,
- *io.portnum=PL_IO_PORT_NUM_1* – powoduje użycie linii portu 1 na płycie



Rys. 8.

ewaluacyjnej połączonej z portem SPI dla modułu GA1000(WiFi),

– *property io.portenabled as byte* – określenie kierunku linii portu (wejście/wyjście),
– *property io.portstate as byte* – zmienna określająca stan portu (H – wysoki/L – niski),

Na początku programu należy również zadeklarować zmienną *i* typu *byte*. Stosujemy prosty zapis: *dim i as byte*. Kompletną procedurę realizującą licznik binarny pokazano na **list. 2**.

Na koniec pozostała nam jeszcze do napisania prosta strona, na której będziemy mogli sygnalizować stan diod statusu SR i SG. Na **list. 3** pokazano kompletny kod źródłowy strony *index.html*. Wymiana danych odbywa się poprzez przesyłanie argumentów funkcji w adresie URL. Jest to metoda GET opisana w standardzie HTML 4.0. W *index.html* nie stykamy się bezpośrednio z platformą sprzętową. Dostęp ten realizuje dopiero *post.html* (**list. 4**).

W celu zaimplementowania kodu źródłowego napisanego w Tibbo Basic w kodzie HTML stosujemy znaczniki <? … ?>. Zarówno wyniki obliczeń, jak i wartości są przesyłane przez zmienne globalne.

Aby móc odczytać zmienne przekazywane z poziomu strony WWW, musimy utworzyć prostą procedurę odczytującą znaczenie przesyłanego łańcucha znaków. Umieszczono ją na **list. 5**.

Gdy już napisaliśmy całe oprogramowanie, przechodzimy do jego testowania. Tibbo IDE dostarcza nam możliwość nie tylko uruchomienia aplikacji, ale również jej debugowania. Debugowanie odbywa się poprzez podglądanie zmiennych i występujących zdarzeń. Istnieje również możliwość ustawiania pułapek w celu chwilowego zatrzymania programu bądź podglądu krokowego.

Aby przetestować napisany program, należy skorzystać z opcji menu *Debug –> Run* bądź nacisnąć *F5*. Jeżeli chcemy podglądać zmienne w czasie działania programu, należy wybrać opcję *Debug –> Add to Watch List*. Pojawi się wtedy nowe okno MDI (**rys. 8**).

Efektem naszej pracy jest zaprogramowany moduł, którym możemy sterować zarówno zdalnie (**rys. 9**), jak i bezpośrednio. Do bezpośredniej interakcji z modułem wykorzystujemy przycisk MD, podłączony do specyficznej dla modułów Tibbo linii Mode. Obsługa zdalna jest realizowana poprzez przesyłanie argumentów w adresie URL metodą GET. Następnie te dane są przetwarzane w ciele programu na informację użyteczną i wykonana zostaje procedura ich obsługi.

List. 3. Strona WWW, index.html

```
<!DOCTYPE HTML PUBLIC „-//W3C//DTD HTML 4.0 Transitional//EN”>
<HTML>
<HEAD>
</HEAD>
<BODY bgcolor=”grey”>
<a href = „http:\\www.soyter.pl” target=_blank><img src = „tibbo.gif”></a>
<b>Kontrolki LED</b>
<br>
<div class=”border1”>
<table border=”0” cellspacing=”0” cellpadding=”0” >
<form action=”post.html” method=”get”>
<tr>
<td>Kolor</td>
<td><input type=”checkbox” name=”red”>Czerwona (R) </td>
<td><input type=”checkbox” name=”green”>Zielona (G)</td>
</tr>
<tr>
<td><input type=”text” name=”pattern” size=”10” class=”input100”></td>
<td><input type=”submit” value=”Enter” tabindex=”2”></td>
</tr>
</table>
<br>
<br>
</form>
</div>
</BODY>
</HTML>
```

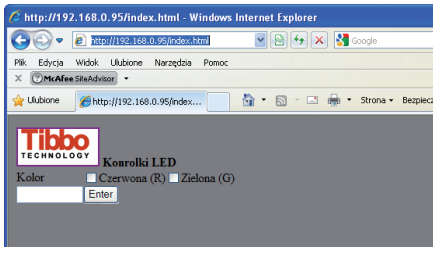
List. 4. Strona WWW – przesyłanie danych, post.html

```
<!DOCTYPE HTML PUBLIC „-//W3C//DTD HTML 4.0 Transitional//EN”>
<HTML>
<HEAD>
<META HTTP-EQUIV=”Refresh”
      CONTENT=”1; URL=index.html”>
</HEAD>
<BODY>
<?  include „global.tbh”
  dim red_on as byte
  dim green_on as byte
  red_on = instr(1,sock.httprqstring,”red=on”,1)
  green_on = instr(1,sock.httprqstring,”green=on”,1)
  pattern = get_http_argument(sock.httprqstring, ”pattern=”)
  if red_on > 0 then
    if green_on > 0 then
      pat.play(„B”,PL_PAT_CANINT)
    else
      pat.play(„R”,PL_PAT_CANINT)
    end if
  else
    if green_on > 0 then
      pat.play(„G”,PL_PAT_CANINT)
    else
      pat.play(„-”,PL_PAT_CANINT)
    end if
  end if
?>
</BODY>
</HTML>
```

Procedura – Metoda GET, main.tbs:

```
'--------------------
'Metoda GET
'--------------------
public function get_http_argument(byref http_req_string as string, byref
argument as string) as string
  dim i, j as byte
  i = instr(1, http_req_string, argument,1)
  if (i = 0) then
    get_http_argument = “”
    exit function
  end if
  i = i + len(argument)
  j = instr(i, http_req_string, “&”,1)
  if (j = 0) then
    j = instr(i, http_req_string, “ “,1)
    if (j = 0) then
      j = len(argument)
    end if
  end if
  get_http_argument = mid(http_req_string, i, j - i)
end function
```

Mikołaj Zarzycki
m.zarzycki@soyter.pl
Soyter Sp. z o.o.



Rys. 9.