Moduły GSM w praktyce (5)

Pierwsze kroki w środowisku OpenAT (ostatni odcinek kursu)

Telefonia GSM w układach automatyki i telemetrii już od kilkunastu lat jest powszechnie wykorzystywana na szeroką skale. Na łamach Elektroniki Praktycznej wielokrotnie publikowano projekty wykorzystujące do komunikacji telefon lub moduł GSM. We wszystkich spotykanych do tej pory rozwiązaniach rolę układu sterującego najczęściej pełnił niezależny mikrokontroler komunikujący się z telefonem/modułem GSM przez port szeregowy wykorzystując odpowiednie komendy AT

List. 4. Kod aplikacji

```
//------------------------------------------------
#include „adl_global.h”

//------------------------------------------------
//Application stack size
const u16 wm_apmCustomStackSize = 2048;
//Flash object name
static const ascii fl_hwnd[] = „EP_Alarm_Flash”;
//Flash object ID
#define TEL_Flash_ID 0
#define CFG_Flash_ID 1
#define MSG_Flash_ID 2
//Global parse buffer
static ascii buf[256];
//Sms handler
static s32 sms_hwnd;
//------------------------------------------------------------
//Konfiguracja diod led
#define IO_COUNT 4

const adl_ioConfig_t io_config[IO_COUNT] =
{
  {ADL_IO_Q2686_GPIO_19, 0, ADL_IO_INPUT,0},
  {ADL_IO_Q2686_GPIO_21, 0, ADL_IO_INPUT,0},
  {ADL_IO_Q2686_GPIO_22, 0, ADL_IO_INPUT,0},
  {ADL_IO_Q2686_GPIO_23, 0, ADL_IO_INPUT,0},
};
//Uchwyt do portu GPIO zawierajacy diody LED
s32 io_hwnd,ioev_hwnd;
//------------------------------------------------
//Callback for allert command
void alarm_callback(adl_atCmdPreParser_t *param)
{
  if(param->Type == ADL_CMD_TYPE_TEST)
  {
    //Test parameter
    adl_atSendResponsePort(ADL_AT_RSP,param->Port,
    „\r\nat+alarm=tel,(R,F,A),msg\r\n”);
  }
  else if(param->Type == ADL_CMD_TYPE_PARA)
  {
    char *p;
    p = ADL_GET_PARAM(param,0);
    adl_flhWrite(fl_hwnd,TEL_Flash_ID,wm_strlen(p)+1,p);
    p = ADL_GET_PARAM(param,1);
    adl_flhWrite(fl_hwnd,CFG_Flash_ID,wm_strlen(p)+1,p);
    p = ADL_GET_PARAM(param,2);
    adl_flhWrite(fl_hwnd,MSG_Flash_ID,wm_strlen(p)+1,p);
    adl_atSendResponsePort(ADL_AT_RSP,param->Port,”\r\nOK\r\n”);
  }
  else if(param->Type == ADL_CMD_TYPE_READ)
  {
    adl_atSendResponsePort(ADL_AT_RSP,param->Port,”\r\nat+alarm=”);
    //Numer telefonu
    s32 len = adl_flhExist(fl_hwnd,TEL_Flash_ID);
    if(len>0)
    {
       adl_flhRead(fl_hwnd,TEL_Flash_ID,len,buf);
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,buf);
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,”,”);
    }
    else
    {
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,”?,”);
```

W poprzednich odcinkach poznaliśmy wszystkie funkcje niezbędne do napisania naszej aplikacji. Jej kod przedstawiono w **list. 4**.

Podobnie jak poprzednio, program rozpoczyna się od wywołania funkcji *adl_main*, w której najpierw jest rejestrowana funkcja obsługi zdarzeń wejścia-wyjścia (*gpio_event_callback*), a następnie za pomocą funkcji *adl_ioSubscribe* konfigurowane są linie IO tak, aby porty GPIO-19, GPIO21, GPIO23 pełniły rolę wejścia. Ponieważ chcemy, aby porty IO były obsługiwane za pomocą funkcji obsługi zdarzenia, musimy do tej funkcji przekazać uchwyt do funkcji obsługi zdarzenia oraz skonfigurować częstotliwość odpytywania za pomocą timera. W naszym przypadku sprawdzanie stanu linii portów IO następuje co 0,1 s. Kolejną czynnością wykonywaną przez funkcję *adl_main* jest rejestracja funkcji obsługi zdarzenia komendy AT+ALARM, (której zadaniem jest zapamiętywanie oraz odczytywanie parametrów konfiguracyjnych) oraz rejestracja chęci skorzystania z pamięci Flash użytkownika za pomocą funkcji *adl_flhSubscribe*, do której przekazujemy identyfikator – klucz w postaci łańcucha tekstowego `fl_hwnd` oraz definiujemy liczbę wpisów w pamięci Flash na 3, tak aby można było zapamiętać nr telefonu, konfigurację oraz treść wiadomości. Po wykonaniu wspomnianych czynności funkcja główna kończy działanie, a cały program jest realizowany przez funkcję obsługi zdarzeń. Teraz, jeżeli w terminalu wpiszemy komendę AT+ALARM z wybranymi parametrami, wywoływana jest funkcja *alarm_callback*, w której jest sprawdzany rodzaj wybranej komendy. W przypadku, gdy jest to zapytanie o dozwolone parametry komendy (ADL_CMD_TYPE_TEST), funkcja zwraca do terminala stosowną odpowiedź i kończy działanie. W przypadku, gdy jest to komenda zlecająca ustawienie parametrów konfiguracyjnych (ADL_CMD_TYPE_PARA), wówczas pobierane są wartości poszczególnych parametrów za pomocą makra ADL_GET_PARAM, a następnie wartości te są zapisywane za pomocą funkcji *adl_flhWrite* pod stosownymi identyfikatorami w wybranym przez nas obszarze pamięci Flash. W przypadku, gdy komendę stanowi zapytanie

o parametry konfiguracyjne, wówczas sprawdzane jest istnienie poszczególnych parametrów za pomocą funkcji *adl_flhExist*, a następnie parametry te są odczytywane za pomocą funkcji *adl_flhRead* oraz wyświetlane w terminalu. W momencie wciśnięcia któregoś z przycisków K1...K4, zostanie wywołane zdarzenie od zmiany stanu linii portów I/O (*gpio_event_callback*). W funkcji tej najpierw z pamięci Flash jest odczytywana zmienna określająca, na jakie zbocze ma reagować nasze urządzenie (narastające, opadające, każde), następnie ze zmiennej `param` odczytywany jest stan oraz numer portu, który wygenerował zdarzenie. Następnie na podstawie poprzedniego stanu linii I/O oraz zmiennej konfiguracyjnej podejmowana jest decyzja czy należy wywołać funkcję *AlarmSms()*, której zadaniem jest wysłanie wiadomości z informacją o alarmie. Funkcja ta jest wywoływana tylko, gdy spełniony jest warunek wynikający ze zbocza sygnału. Zadaniem funkcji *AlarmSms* jest sformatowanie wiadomości tekstowej informującej o stanie poszczególnych linii oraz wysłanie wiadomości tekstowej pod numer znajdujący się w pamięci konfiguracyjnej. W funkcji tej najpierw z pamięci Flash jest odczytywany nagłówek wiadomości tekstowej, następnie na podstawie stanu poszczególnych linii I/O formatowana jest treść wiadomości informująca o stanie poszczególnych linii. Na zakończenie odczytywany jest z pamięci Flash numer telefonu, a następnie z bufora wiadomości wysyłana jest wiadomość tekstowa pod wskazany numer, po czym funkcja kończy działanie. Teraz w momencie wysłania wiadomości lub wystąpienia błędu wywoływana jest funkcja obsługi zdarzenia SMS (*sms_ctrl_callback*), której jedynym zadaniem jest wypisanie w terminalu informacji o tym, czy udało się wysłać wiadomość tekstową pod wskazany numer telefonu.

Komunikacja GSM – sterowanie urządzeniem za pomocą SMS-ów

Nauczyliśmy się wcześniej wysyłać wiadomości tekstowe, w sytuacji awaryjnej, którą była określona zmiana stanu linii wejściowych. Jednak w praktyce często zachodzi potrzeba sterowania jakimiś urządzeniami zdalnie za pomocą SMS-ów. Na przykład chcemy z odpowiednim wyprzedzeniem włączać ogrzewanie domku letniskowego przed naszym przyjazdem do niego. Nauczymy się teraz, w jaki sposób możemy sterować liniami wyjściowymi za pomocą SMS-ów oraz odczytywać stan poszczególnych linii na żądanie. Napiszemy prosty program, który w momencie odebrania SMS-a o określonej treści będzie sterował diodami LED na płycie prototypowej lub odczytywał stany wejść. Nic nie stoi na przeszkodzie, aby do linii portów I/O zamiast diod LED podłączyć jakiś układ wykonawczy z przekaźnikiem, za pomocą którego będziemy sterować jakimś większym urządzeniem. Jeżeli zostanie odebrany SMS o treści *?*, co oznacza zapytanie o stany wejść, wówczas program odczyta stany 4 wejść GPIO19, 21, 22, 23 i odeśle do

List. 4. c.d.

```
    }
    //Configuration
    len = adl_flhExist(fl_hwnd,CFG_Flash_ID);
    if(len>0)
    {
       adl_flhRead(fl_hwnd,CFG_Flash_ID,len,buf);
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,buf);
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,",");
    }
    else
    {
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,"?,");
    }
    //Message
    len = adl_flhExist(fl_hwnd,MSG_Flash_ID);
    if(len>0)
    {
       adl_flhRead(fl_hwnd,MSG_Flash_ID,len,buf);
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,buf);
       adl_atSendResponsePort(ADL_AT_RSP,param->Port,"\r\n");
     }
     else
     {
        adl_atSendResponsePort(ADL_AT_RSP,param->Port,"?\r\n");
     }
  }
}
//-------------------------------------------------
void AlarmSMS(u8 t,s32 gpio,adl_ioRead_t* p,s32 size)
{
   ascii smst[161];
   int i,len;
   len = adl_flhExist(fl_hwnd,MSG_Flash_ID);
   if(len>0)
   {
      adl_flhRead(fl_hwnd,MSG_Flash_ID,len,buf);
   }
   else
   {
      wm_strcpy(buf,"No msg");
   }
   buf[100]=0;
   for(i=0;i<size;i++)
   {
      if(p[i].eState) wm_strcat(&buf[100],"ON „);
      else wm_strcat(&buf[100],"OFF „);
   }
   wm_sprintf(smst,"%s Typ %c Port %d IO: %s\r\n",buf,t,gpio-ADL_IO_Q2686_
GPIO_1+1,&buf[100]);
   adl_atSendResponse(ADL_AT_UNS,smst);
   //Message
   len = adl_flhExist(fl_hwnd,TEL_Flash_ID);
   if(len>0)
   {
      adl_flhRead(fl_hwnd,TEL_Flash_ID,len,buf);
   }
   else
   {
      return;
   }
   int res = adl_smsSend(sms_hwnd,buf,smst,ADL_SMS_MODE_TEXT);
   wm_sprintf(buf,"\r\nSms send with result %d\r\n",res);
   adl_atSendResponse(ADL_AT_UNS,buf);
}
//-------------------------------------------------
void gpio_event_callback(s32 gpio_hwnd,adl_ioEvent_e event,u32 size,void
*param)
{
   static u8 io_p[4];
   int i;
   //Jezeli to nie jest zmiana stanu to wyjdz
   if(event!=ADL_IO_EVENT_INPUT_CHANGED) return;
 /*
   wm_sprintf(buf,"\r\nGPIO %d new value %d size %d\r\n",((adl_ioRead_
t*)param)[0].eLabel,
      ((adl_ioRead_t*)param)[0].eState,size);
   adl_atSendResponse(ADL_AT_UNS,buf);
   wm_sprintf(buf,"\r\nGPIO %d new value %d size %d\r\n",((adl_ioRead_
t*)param)[1].eLabel,
      ((adl_ioRead_t*)param)[1].eState,size);
   adl_atSendResponse(ADL_AT_UNS,buf);
   wm_sprintf(buf,"\r\nGPIO %d new value %d size %d\r\n",((adl_ioRead_
t*)param)[2].eLabel,
      ((adl_ioRead_t*)param)[2].eState,size);
   adl_atSendResponse(ADL_AT_UNS,buf);
   wm_sprintf(buf,"\r\nGPIO %d new value %d size %d\r\n",((adl_ioRead_
t*)param)[3].eLabel,
      ((adl_ioRead_t*)param)[3].eState,size);

   adl_atSendResponse(ADL_AT_UNS,buf); */
   //Pobierz rodzaj alarmu
   //Configuration
   int len = adl_flhExist(fl_hwnd,CFG_Flash_ID);
   //Urzadzenie nie skonfigrurowane
   if(len<=0) return;
   adl_flhRead(fl_hwnd,CFG_Flash_ID,len,buf);
   u8 al_type = buf[0];
   int alarm = 0;
   for(i=0;i<4;i++)
   {
    switch(al_type)
    {
     //Zbocze narastajace
```

List. 4. c.d.

```
      case 'r':
      case ,R':
            if(!io_p[i] && ((adl_ioRead_t*)param)[i].eState )
            alarm = ((adl_ioRead_t*)param)[i].eLabel.GenericLabel;
            break;
            //Zbocze opadajace
      case ,f':
      case ,F':
            if(io_p[i] && !((adl_ioRead_t*)param)[i].eState )
            alarm=((adl_ioRead_t*)param)[i].eLabel.GenericLabel;
            break;
          //Dowolne zbocze
      case 'A':
      case ,a':
           alarm = ((adl_ioRead_t*)param)[i].eLabel.GenericLabel;
           break;
     }
    }

    //Zapisz poprzedni stan
    for(i=0;i<3;i++)
    {
      io_p[i] = ((adl_ioRead_t*)param)[i].eState;
    }
    //Jezeli alarm to generuj
    if(alarm)
    {
       AlarmSMS(al_type,alarm,(adl_ioRead_t*)param,size);
    }
}

//------------------------------------------------
//Sms handler function
bool sms_msg_callback(ascii *tel,ascii *time,ascii *text)
{
   //Forward sms to ADL
   return TRUE;
}

//------------------------------------------------
//Sms control handler
void sms_ctrl_callback(u8 event,u16 nb)
{
   switch(event)
   {
    case ADL_SMS_EVENT_SENDING_OK:
        adl_atSendResponse(ADL_AT_UNS,"\r\nOK send alert sms\r\n");
        break;
    case ADL_SMS_EVENT_SENDING_ERROR:
        adl_atSendResponse(ADL_AT_UNS,"\r\nERROR send alert sms\r\n");
        break;
   }
}
//------------------------------------------------
void adl_main ( adl_InitType_e  InitType )
{
   //Subscribe gpio
   ioev_hwnd = adl_ioEventSubscribe(gpio_event_callback);
   if(ioev_hwnd>=0)
   {
      adl_atSendResponse(ADL_AT_UNS,"\r\nGoSmsAlert OK\r\n");
   }
   else
   {
      adl_atSendResponse(ADL_AT_UNS,"\r\nGoSmsAlert FAIL\r\n");
   }
   io_hwnd = adl_ioSubscribe(4,io_config,ADL_TMR_TYPE_100MS,1,ioev_hwnd);
   if(io_hwnd<0)
   {
      adl_atSendResponse(ADL_AT_UNS,"\r\nGoSmsAlert FAIL3 \r\n");
      wm_sprintf(buf,"Error %d\n",io_hwnd);
      adl_atSendResponse(ADL_AT_UNS,buf);
   }
    adl_atCmdSubscribe(„at+alarm",alarm_callback,ADL_CMD_TYPE_TEST|
ADL_CMD_TYPE_PARA|ADL_CMD_TYPE_READ|0x33);
    adl_flhSubscribe(fl_hwnd,MSG_Flash_ID+1);
    //Subscribe Sms service
    sms_hwnd = adl_smsSubscribe(sms_msg_callback,sms_ctrl_callback,ADL_SMS_
MODE_TEXT);
}
//------------------------------------------------
```

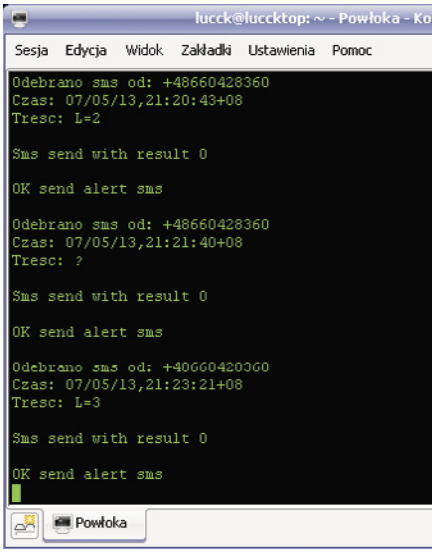
List. 5. Przykładowy program

```
//------------------------------------------------------------
#include "adl_global.h"
//------------------------------------------------------------
const u16 wm_apmCustomStackSize = 1024;

//------------------------------------------------------------
//Konfiguracja diod led
#define LEDS_COUNT 2

const adl_ioConfig_t led_config[LEDS_COUNT] =
{
  {ADL_IO_Q2686_GPIO_15, 0, ADL_IO_OUTPUT, ADL_IO_HIGH},
  {ADL_IO_Q2686_GPIO_16, 1, ADL_IO_OUTPUT, ADL_IO_HIGH},
};

//Uchwyt do portu GPIO zawierajacy diody LED
s32 led_hwnd;
```



Rys. 19. Widok okna terminala w momencie obsługi zdarzeń od SMS-ów

użytkownika zwrotnego SMS-a o następującej treści: „0-ON 1-OFF 2-OFF 3-OFF" podając stan poszczególnych linii IO. W przypadku, gdy użytkownik wyśle SMS-a o treści: L=n, gdzie n jest odpowiednio liczbą z zakresu 0...3, wówczas na płytce prototypowej zostaną włączone lub wyłączone diody RXD i CTS w zależności od stanu bitów liczby (*n*), a do użytkownika zostanie odesłana wiadomość informująca o bieżącym stanie diod LED w następującej postaci: np. „Diody led zostały ustawione LED1=WYL LED2=ZAL". Jak więc widzimy działanie tej aplikacji jest bardzo proste i nie wykorzystuje ona żadnych danych konfiguracyjnych. Należy jednak pamiętać, że aplikacja ta nie zapewnia żadnej kontroli użytkownika, więc dowolna nieuprawniona osoba może sterować naszymi urządzeniami. Należy to mieć na uwadze i podczas pisania właściwego programu uzupełnić kod chociażby o sprawdzanie numery telefonu nadawcy, tak aby tylko osoba uprawniona mogła wysyłać rozkazy. Program oprócz wysyłania wiadomości tekstowych do użytkownika wyświetla w terminalu wszystkie zdarzenia oraz informacje o statusie wysyłania SMS-a. Widok terminala w momencie obsługi zdarzeń od SMS-ów przedstawiono na **rys. 19**.

Wszystkie potrzebne funkcje niezbędne do napisania tego programu poznaliśmy w poprzednich odcinkach artykułu, jedynie pewnego

czytaj na stronie 102

omówienia wymaga funkcja `bool sms_msg_callback(ascii *tel, ascii* time, ascii *text);`, która jest wywoływana w momencie odebrania wiadomości tekstowej. Funkcja ta powinna być wcześniej zarejestrowana za pomocą *adl_smsSubscribe()*. Do funkcji tej przekazywane są następujące argumenty:
`tel` – Numer telefonu w postaci tekstowej, z którego została wysłana wiadomość.
`time` – Czas nadejścia wiadomości tekstowej.
`text` – Treść odebranej wiadomości.

Funkcja ta powinna zwracać wartość TRUE, jeżeli chcemy, aby odebrana wiadomość była przekazana do dalszego przetwarzania przez system operacyjny modułu. Opcja ta jest użyteczna w przypadku, gdy nie chcemy odbierać wiadomości. Wówczas deklarujemy sobie pustą funkcję zwracającą wartość TRUE (tak jak w poprzednim przykładzie). Jeżeli funkcja zwraca wartość FALSE, wówczas odebrana wiadomość nie jest przekazywana do przetwarzania przez system operacyjny i funkcja powinna sama przetworzyć odebranego SMS-a. Reasumując, w momencie wywołania zdarzenia od odebrania SMS, za pomocą tej funkcji możemy w naszej aplikacji odczytać treść odebranego SMS-a. W poprzednim programie do odczytywania stanu linii portu wykorzystywaliśmy funkcję obsługi zdarzeń, która była wywoływana w momencie zmiany stanu na liniach IO. Istnieje również alternatywna metoda polegająca na odczytaniu stanu linii wejściowych na żądanie. Metoda ta ma tę zaletę, że nie wymaga konieczności użycia dodatkowego Timera. Odczyt stanu linii portu IO na żądanie umożliwia funkcja `s32 adl_ioRead(s32 GpioHandle, u32 GpioNb, adl_ioRead_t *GpioRead);`, która przyjmuje następujące parametry:
`GpioHandle` – Uchwyt do linii portów I/O.
`GpioNb` – Liczba wejść, których stan chcemy odczytać.
`GpioRead` – Wskaźnik do omawianej wcześniej tablicy struktur zawierających informację o stanie wejść. Przed wywołaniem należy ją wypełnić identyfikatorami linii I/O, które chcemy odczytać. Po zakończeniu funkcja uzupełnia tę strukturę o stan poszczególnych linii. Funkcja zwraca wartość OK w przypadku, gdy operacja przebiegła pomyślnie, w przeciwnym przypadku natomiast zwracany jest kod informujący o błędzie.

Po uzupełnieniu wiadomości na temat funkcji API modułu przejdziemy teraz do omówienia programu, którego kod został przedstawiony na **list. 5**.

Program rozpoczyna wykonanie od funkcji *adl_main*, w której najpierw są inicjalizowane porty obsługujące diody LED oraz porty obsługujące linie wejściowe. Na zakończenie rejestrowane są funkcje obsługi zdarzeń dotyczących SMS-ów, po czym funkcja inicjalizacyjna kończy działanie. Od tego momentu działanie całego programu zależy od funkcji obsługi zdarzeń. W momencie odebrania wiadomości tekstowej z sieci wywoływana jest funkcja *sms_msg_callback()*, w której najpierw na konsoli jest wypisywana treść odebranego SMS-a, a następnie

List. 5. c.d.

```
//Konfiguracja wejsc/wyjsc
#define IO_COUNT 4

const adl_ioConfig_t io_config[IO_COUNT] =
{
  {ADL_IO_Q2686_GPIO_19, 0, ADL_IO_INPUT,0},
  {ADL_IO_Q2686_GPIO_21, 0, ADL_IO_INPUT,0},
  {ADL_IO_Q2686_GPIO_22, 0, ADL_IO_INPUT,0},
  {ADL_IO_Q2686_GPIO_23, 0, ADL_IO_INPUT,0},
};


//Uchwyd do portu GPIO zawierajacy diody LED
s32 io_hwnd;

//-------------------------------------------------------------
//Global parse buffer
static ascii buf[256];
//Sms handler
static s32 sms_hwnd;
//-------------------------------------------------------------
#define LED1_BIT 1
#define LED2_BIT 2

//Sterowanie diodami led
void set_led(u8 leds)
{
   if(leds & 1) adl_ioWriteSingle(led_hwnd,ADL_IO_Q2686_GPIO_15,ADL_IO_LOW);
   else adl_ioWriteSingle(led_hwnd,ADL_IO_Q2686_GPIO_15,ADL_IO_HIGH);
   if(leds & 2) adl_ioWriteSingle(led_hwnd,ADL_IO_Q2686_GPIO_16,ADL_IO_LOW);
   else adl_ioWriteSingle(led_hwnd,ADL_IO_Q2686_GPIO_16,ADL_IO_HIGH);
}

//-------------------------------------------------------------
//Pobierz stan wejsc IO
u8 get_input(void)
{
  u8 retv=0;
  adl_ioRead_t inp[4];
  inp[0].eLabel.Q2686_Label = ADL_IO_Q2686_GPIO_19;
  inp[1].eLabel.Q2686_Label = ADL_IO_Q2686_GPIO_21;
  inp[2].eLabel.Q2686_Label = ADL_IO_Q2686_GPIO_22;
  inp[3].eLabel.Q2686_Label = ADL_IO_Q2686_GPIO_23;
  adl_ioRead(io_hwnd,4,&inp);
  int i;
  for(i=0;i<3;i++)
  {
    if(inp[i].eState) retv |= 1<<i;
  }
  return retv;
}
//-------------------------------------------------
//Sms handler function
bool sms_msg_callback(ascii *tel,ascii *time,ascii *text)
{
  adl_atSendResponse(ADL_AT_UNS,"\r\nOdebrano sms od: ");
  adl_atSendResponse(ADL_AT_UNS,tel);
  adl_atSendResponse(ADL_AT_UNS,"\r\nCzas: ");
  adl_atSendResponse(ADL_AT_UNS,time);
  adl_atSendResponse(ADL_AT_UNS,"\r\nTresc: ");
  adl_atSendResponse(ADL_AT_UNS,text);
  adl_atSendResponse(ADL_AT_UNS,"\r\n");
  //No Forward sms to ADL
  if(text[0]=='?')
  {
     //Get input state
     wm_strcpy(buf,"Biezacy stan wejsc: ");
     u8 v;
     v = get_input();
     int i;
     for(i=0;i<4;i++)
     {
       if(v & (1<<i)) wm_sprintf(&buf[200],"%d-%s ",i,"ON");
       else wm_sprintf(&buf[200],"%d-%s ",i,"OFF");
       wm_strcat(buf,&buf[200]);
     }
  }
  else if( (text[0]=='l' || text[0]=='L') && text[1]=='=')
  {
    int val;
    val = wm_atoi(&text[2]);
    if(val>3 || val<0)
    {
       wm_strcpy(buf,"Bledna komenda dozwolone wartosci dla diod led
l=0,1,2,3");
    }
    else
    {
       wm_strcpy(buf,"Diody led zostaly ustawione: LED1=");
       if(val&1) wm_strcat(buf,"ZAL ");
       else wm_strcat(buf,"WYL ");
       wm_strcat(buf,"LED2=");
       if(val&2) wm_strcat(buf,"ZAL ");
       else wm_strcat(buf,"WYL ");
       set_led(val);
    }
  }
  else
  {
     //Calkiem nieznana komenda
     wm_strcpy(buf,"LBRPT: Nieznana komenda: Znane komendy ? lub L=0,1,2,3");
  }
  int res = adl_smsSend(sms_hwnd,tel,buf,ADL_SMS_MODE_TEXT);
```

List. 5. c.d.

```
  wm_sprintf(buf,"\r\nSms send with result %d\r\n",res);
  adl_atSendResponse(ADL_AT_UNS,buf);
  return FALSE;
}

//------------------------------------------------
//Sms control handler
void sms_ctrl_callback(u8 event,u16 nb)
{
  switch(event)
  {
   case ADL_SMS_EVENT_SENDING_OK:
        adl_atSendResponse(ADL_AT_UNS,"\r\nOK send alert sms\r\n");
        break;
   case ADL_SMS_EVENT_SENDING_ERROR:
        adl_atSendResponse(ADL_AT_UNS,"\r\nERROR send alert sms\r\n");
        break;
  }
}

//------------------------------------------------
//Funkcja glowna programu
void adl_main ( adl_InitType_e  InitType )
{
  led_hwnd = adl_ioSubscribe(LEDS_COUNT,led_config,0,0,0);
  io_hwnd = adl_ioSubscribe(IO_COUNT,io_config,0,0,0);
  //Subscribe Sms service
  sms_hwnd = adl_smsSubscribe(sms_msg_callback,sms_ctrl_callback,ADL_SMS_MODE_
TEXT);
}

//------------------------------------------------------------
```

jest sprawdzany pierwszy bajt odebranej wiadomości. Jeżeli pierwszy znak zawiera wartość *?*, oznacza to odebranie zapytania o stan linii I/O, natomiast odebranie znaku *L* oznacza odebranie rozkazu załączenia diod LED. W przypadku, gdy odebrano zapytanie o stan linii wejściowych, wówczas jest odczytywany stan tych linii, a następnie za pomocą funkcji *wm_sprintf* oraz *wm_strcat* formatowana jest treść wiadomości zawierająca informację o stanie wejść. W przypadku, gdy odebrano rozkaz włączenia diod LED, sprawdzany jest prawidłowy nagłówek wiadomości L=, a następnie wartość określająca stan diod LED jest zamienia na liczbę. Jeżeli liczba ta jest większa od 3, wówczas jest tworzony komunikat o błędnej wartości, natomiast jeżeli treść rozkazu jest prawidłowa, wówczas w zależności od stanu poszczególnych bitów są włączane lub wyłączane diody LED. Na zakończenie funkcji przygotowana w buforze wiadomość tekstowa jest wysyłana za pomocą funkcji *adl_smsSend*. Teraz jeżeli uda się wysłać wiadomość lub wystąpił błąd wywoływana jest funkcja *sms_ctrl_callback*, której zadaniem jest jedynie wypisanie w terminalu statusu wysyłania wiadomości tekstowej.

Zakończenie

Zaprezentowany na łamach niniejszego cyklu moduł WaveCom Q2686, umożliwia bardzo wygodne tworzenie własnych urządzeń nie posiadających odrębnego procesora, ponieważ program użytkownika jest wykonywany przez moduł GSM. Dzięki takiemu podejściu możemy zmniejszyć rozmiar płytki PCB, przez co nasze urządzenie staje się mniejsze i tańsze. Możemy także zaoszczędzić wiele czasu, ponieważ środowisko programistyczne OpenAT zawiera szereg funkcji ułatwiających pracę, jak chociażby automatyczne przetwarzanie komend AT. Nie ma jednak róży bez kolców: pisząc własne oprogramowanie wykorzystujące moduł Q2686 musimy ściśle dostosować się do reguł narzuconych przez środowisko OpenAT. Projekty wymagające specjalnych funkcji lub zależności czasowych i tak będą więc wymagały użycia niezależnego procesora służącego do sterowania. W 90% przypadków jednak oprogramowanie wykonywanie w module będzie spełniać oczekiwania użytkownika. Z uwagi na ograniczone łamy niniejszego cyklu, zapoznano tutaj Czytelnika tylko z najbardziej podstawową funkcjonalnością środowiska OpenAT, oraz prostymi programami przykładowymi, których celem było pokazanie, w jaki sposób tworzyć aplikację. Osoby zainteresowane poznaniem szerszych zagadnień dotyczących niniejszej tematyki będą musiały sięgnąć do dokumentacji dostarczanej przez producenta modułu. Mam nadzieje, że dzięki temu krótkiemu cyklowi udało mi się wprowadzić Czytelnika w świat środowiska OpenAT.

Niniejszy cykl artykułów został przygotowany z wykorzystaniem wolnego oprogramowania OpenOffice.org. oraz systemu operacyjnego Ubuntu Linux (*www.ubuntu.com*). System operacyjny Windows, na którym działał OpenAT został uruchomiony w oprogramowaniu wirtualizacyjnym **KVM** (Kernel Based Virtual Machine) dostępnym na licencji GPL.

Lucjan Bryndza SQ7FGB, EP
lucjan.bryndza@ep.com.pl