Moduły GSM w praktyce (4)

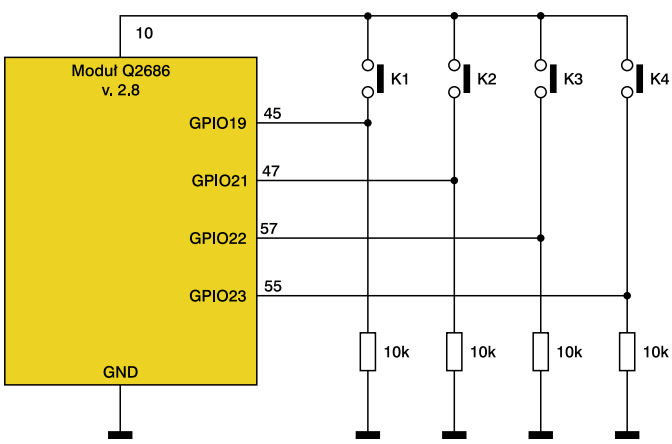
Pierwsze kroki w środowisku OpenAT

Telefonia GSM w układach automatyki i telemetrii już od kilkunastu lat jest powszechnie wykorzystywana na szeroką skalę. 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.

Komunikacja w sieci GSM

W tym odcinku kursu omówimy wysyłanie SMS-ów, czyli program monitorujący stan określonych linii i wysyłający SMS-y w przypadku zmiany tych linii.

Po zapoznaniu się z podstawową funkcjonalnością modułów GSM, przyszła pora, aby wykorzystać moduł WaveCom w zastosowaniu, do jakiego został on właściwie stworzony, czyli do komunikacji w sieci GSM. Bardzo często podczas pracy z modułami GSM zachodzi konieczność wysyłania informacji o wystąpieniu sytuacji awaryjnej (alarm). Do tego celu świetnie nadają się SMS–y, które w dyskretny sposób informują użytkownika o zajściu jakiegoś zdarzenia. Na przykład zmiana stanu określonych linii modułu może informować o przekroczeniu dozorowanej strefy lub informować o przekroczeniu parametrów jakiegoś procesu przemysłowego wymagającego ingerencji użytkownika. W bieżącym odcinku napiszemy aplikację, która w momencie określonej zmiany stanu wybranych linii I/O spowoduje wysłanie SMS–a z alarmem zawierającym wybrany komunikat tekstowy oraz informację o liniach, które wywołały alarm. Przyjmiemy tutaj założenie, że jako linie powodujące wystąpienie alarmu wykorzystamy porty GPIO19, GPIO21, GPIO22, GPIO23 pracujące w standardzie 2,8 V. Niestety tym razem nie obejdzie się bez zabawy z lutownicą, ponieważ do wyżej wspomnianych linii będziemy musieli przylutować prosty układ z przełącznikami DIP–switch, którego schemat przedstawiono na **rys. 17.** Wciśnięcie wybranego przycisku, będzie powodowało zmianę stanu odpowiadającej mu linii, będziemy więc mieli możliwość symulacji zgłaszania sytuacji alarmowych za pomocą przycisków. Za pomocą odpowiedniej komendy konfiguracyjnej będziemy mogli wybrać czy układ ma reagować na zbocze opadające sygnału, zbocze narastające, czy na dowolną zmianę stanu linii. Używając odpowiednich komend będziemy również mogli zmienić numeru telefonu, pod który będzie wysyłany alarm, a także zdefiniować treść tego alarmu. Tym razem wszystkie ustawienia konfiguracyjne o których wspomniano będą przechowywane w nieulotnej pamięci Flash dla danych. Dzięki temu moduł będzie pamiętał wszystkie ustawienia konfiguracyjne nawet po wyłączeniu napięcia zasilania. Do konfiguracji wszystkich wybranych ustawień służy jedna zdefiniowana przez nas komenda AT+ALARM, która przyjmuje trzy parametry: numer telefonu, rodzaj zbocza generującego alarm oraz treść komunikatu. W **tab. 7** przedstawiono rozkazy oraz argumenty, jakie może przyjmować zdefiniowana przez naszą aplikację komenda AT+ALARM.



Rys. 17. Schemat prostego układu z przełącznikami DIP–switch do symulacji alarmu

Jak więc widzimy, nasza komenda konfiguracyjna jest bardzo prosta, a zarazem użyteczna, ponieważ możemy skonfigurować wybrane parametry według naszych potrzeb, bez sztywnego definiowania ich w programie. Dodatkowo parametry te nie są tracone po wyłączeniu zasilania modułu. Na przykład: aby skonfigurować nasz moduł tak, aby w momencie wystąpienia zbocza narastającego na dowolnym z wejść został wysłany SMS o treści „Uwaga! Wystąpił alarm w maszynie" pod numer 660428360, należy z terminala wydać komendę `AT+ALARM="+48660428360","R","Uwaga! Wystąpił alarm w maszynie"` i nacisnąć klawisz ENTER. W tym momencie moduł powinien przyjąć komendę odpowiadając `OK`. Od tej pory w momencie wciśnięcia dowolnego przycisku (K1...K4), pod wskazany numer telefonu zostanie wysyłany SMS o treści: Uwaga! Wystąpił alarm w maszynie Typ R Port 21 IO: OFF ON OFF OFF". Oznacza to, że wystąpienie alarmu spowodowała linia GPIO21, a stan wszystkich linii w kolejności jest następujący: GPIO19 – OFF, GPIO21 – ON, GPIO22 – OFF, GPIO23 – ON. Ponadto treść tej wiadomości zostanie wyświetlona na terminalu oraz dodatkowo znajdzie się tam również informacja o tym, czy udało się wysłać wiadomość pod wskazany numer. Widok terminala w momencie wystąpienia alarmu przedstawiono na **rys. 18.**

Analizując ten program nauczymy się kilku nowych rzeczy, mianowicie poznamy: w jaki sposób zapisywać i odczytywać dane z nieulotnej pamięci Flash modułu, wysyłać wiadomości SMS oraz wykorzystywać zdarzenia generowane w momencie zmiany stanu określonych linii portów I/O. Do obsługi pamięci danych użytkownika służy odpowiedni zestaw funkcji API modułu. Pamięć tę możemy wykorzystywać podobnie, jak zwykłą pamięć EEPROM mikrokontrolerów, jednak sposób jej obsługi jest zdecydowanie wygodniejszy niż w przypadku pamięci wspomnianej wcześniej. Aby skorzystać z nieulotnej pamięci danych modułu, należy wcześniej zadeklarować chęć jej użycia za pomocą funkcji *adl_flhSubscribe*, która przyjmuje następujące parametry oraz zdefiniowana jest następująco:

```
s8 adl_flhSubscribe(ascii *Handle, u16 NbObjectRes);
```

Tab. 7. Rozkazy oraz argumenty przyjmowane przez komendę AT+ALARM

Komenda	Opis
AT+ALARM=tel,zbocze,komunikat	Komenda powoduje ustawienie oraz zapisanie w nieulotnej pamięci Flash konfiguracji programu. Parametr `tel.` powinien zawierać numer telefonu, pod który będą wysyłane wiadomości tekstowe z alarmami. Parametr `zbocze` umożliwia określenie czy SMS z alarmem będzie generowany w momencie wystąpienia zbocza narastającego (`R`), zbocza opadającego (`F`) lub w przypadku wystąpienia dowolnego zbocza (`A`). Ostatni, trzeci parametr `komunikat` powinien zawierać komunikat, jaki zostanie wysłany w momencie wystąpienia alarmu
AT+ALARM?	Odczytuje i wyświetla na terminalu parametry konfiguracyjne w postaci AT+ALARM=(tel,zbocze,wiadomość), co pozwala sprawdzić parametry konfiguracyjne, jakie zostały wpisane do pamięci Flash modułu
AT+ALARM=?	Wyświetla na terminalu dostępne opcje konfiguracyjne naszej komendy

`Handle` – Parametr ten powinien zawierać wskaźnik do łańcucha tekstowego, który służy do identyfikacji obszaru pamięci Flash aplikacji. W zasadzie może to być dowolny łańcuch tekstowy jednoznacznie identyfikujący nasz obszar pamięci, na przykład: „Moje obiekty programu".

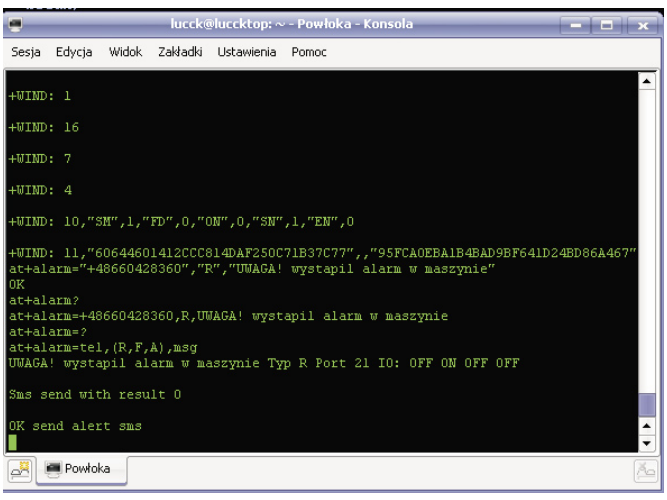
`NbObjectRes` – Parametr ten powinien zawierać maksymalną liczbę obiektów, jakie będziemy przechowywać w naszym obszarze. Na przykład wpisanie wartości 10 spowoduje, że wskazany przez nas obszar pamięci Flash będzie mógł przechowywać 10 identyfikatorów obiektów.

Opisywana wyżej funkcja w przypadku powodzenia zwraca OK, natomiast w przeciwnym przypadku zwracany jest kod błędu. W przypadku, gdy funkcja ta zwróci błąd możemy spróbować skasować cały obszar pamięci Flash użytkownika za pomocą komendy AT+WOPEN=3, a następnie spróbować wywołać ponownie tę funkcję. Sytuacja taka ma miejsce, gdy przy pierwszym użyciu tej funkcji z danym identyfikatorem podamy mniejszą liczbę obiektów niż przy drugim wywołaniu. Jednym sposobem na wyjście z tej sytuacji jest wówczas skasowanie obszaru pamięci użytkownika za pomocą komendy AT+WOPEN=3.

Po zadeklarowaniu chęci użycia danego obszaru pamięci możemy go zapisywać za pomocą funkcji *adl_flhWrite*:

```
s8 adl_flhWrite(ascii *Handle, u16 ID, u16 Len, u8
*WriteData);
```

`Handle` – Parametr ten powinien zawierać wskaźnik do łańcucha tekstowego, który służy do identyfikacji obszaru pamięci Flash aplikacji. W zasadzie może to być dowolny łańcuch tekstowy jednoznacznie identyfikujący nasz obszar pamięci na przykład: „Moje obiekty programu".



Rys. 18. Okno terminala w momencie wystąpienia alarmu

`ID` – Identyfikator obiektu, którego maksymalna dozwolona wartość powinna być mniejsza o 1 od liczby obiektów zadeklarowanych za pomocą wcześniej opisanej funkcji. Identyfikator ten pozwala jednoznacznie określić numer obiektu zawierającego konkretne dane.

`WriteData,Len` – Wskaźnik do bufora z danymi do zapisania na określonej pozycji obiektu (ID) oraz długość tego bufora.

W przypadku pomyślnego zapisu danych do pamięci Flash funkcja zwraca wartość OK, natomiast w przeciwnym przypadku zwracany jest kod błędu. Odczyt zapisanego wcześniej obszaru pamięci Flash umożliwia funkcja:

```
s8 adl_flhWrite(ascii *Handle, u16 ID, u16 Len, u8
*ReadData);
```

Funkcja ta przyjmuje identyczne parametry jak poprzednio, przy czym w tym przypadku dane są kopiowane z obszaru pamięci Flash oznaczonego uchwytem `Handle` i identyfikatorem `ID` do bufora przekazanego do zmiennej `ReadData`. Aby odczytać wybrany obszar pamięci musimy niestety znać jego długość. Odczytanie długości danych wybranego obszaru pamięci Flash zapewnia funkcja `s8 adl_flhExist(ascii *Handle, u16 ID)`, która na podstawie uchwytu (*Handle*) oraz identyfikatora obiektu (*ID*) odczytuje wielkość tego obszaru, natomiast w przypadku jego braku zwraca 0 lub kod błędu, który jest wartością ujemną.

Wykorzystując ten zestaw funkcji mamy możliwość zapisywania dowolnych zmiennych w nieulotnej pamięci modułu. W naszym przypadku będziemy tutaj zapisywać poszczególne parametry przekazane komendzie AT+ALARM.

Do tej pory porty I/O wykorzystywaliśmy tylko do sterowania diodami. Obecnie poznamy, w jaki sposób możemy wykorzystywać porty I/O jako wejścia w taki sposób, aby w momencie zmiany stanu dowolnie wybranych linii portu wywoływana była funkcja informująca o tym zdarzeniu. Aby stworzyć funkcję obsługi zdarzenia reagującego na zmianę stanów linii I/O, musimy najpierw zarejestrować funkcję obsługi zdarzeń od linii wejścia–wyjścia za pomocą funkcji:

```
s32 adl_ioEventSubscribe(adl_ioHdlr_f GpioEventHan-
dler);
```

Funkcja ta przyjmuje wskaźnik do funkcji obsługi zdarzeń I/O oraz zwraca uchwyt obsługi tego zdarzenia, albo błąd, który jest wartością ujemną. Funkcja obsługi zdarzenia IO powinna być zdefiniowana następująco:

```
void gpio_event_callback(s32 gpio_hwnd,adl_ioEvent_e
event,u32 size,void *param);
```

Zwrócony przez tę funkcję uchwyt obsługi zdarzenia musi być następnie przekazany do omawianej w 3 części artykułu funkcji *adl_ioSubscribe*. W funkcji tej należy także wypełnić pola dotyczące rodzaju wykorzystywanego timera oraz częstotliwości sprawdzania wejść. Poza tym, aby zdarzenie było wywołane, chociaż jedna linia I/O musi być zdefiniowana jako wejściowa. Po wywołaniu funkcji *adl_ioSubscribe* w momencie zmiany stanu dowolnej linii I/O zostanie wywołana zarejestrowana wcześniej funkcja obsługi zdarzenia, której parametry mają następujące znaczenie:

`gpio_hwnd` – Parametr ten zawiera uchwyt do linii portów I/O

`event` – Typ zdarzenia które spowodowało wywołanie funkcji. W tym przypadku interesuje nas tylko wartość `ADL_IO_EVENT_INPUT_CHANGED`, w pozostałych przypadkach możemy opuścić funkcję.

`param,size` – Parametr ten zawiera wskaźnik na dane związane z określonym zdarzeniem, w przypadku `ADL_IO_EVENT_INPUT_CHANGED` zmienną tę, zawierającą informację o tym, które z wejść uległy zmianie, powinniśmy rzutować na typ `adl_ioRead_t*`. Struktura ta jest zdefiniowana następująco:

```
typedef struct
{
  adl_ioLabel_u eLabel;
  adl_ioState_e eState;
} adl_ioRead_t;
```

Pole `eLabel` zawiera numer portu I/O, który uległ zmianie. W przypadku modułu Q2686 pole to może przyjąć wartość od `ADL_IO_Q2686_`

GPIO_1 do ADL_IO_Q2686_GPIO_44. Pole eState zwraca natomiast stan, jaki przyjęła wybrana linia, gdzie dozwolone wartości to odpowiednio 0 lub 1. Badając zawartość tej struktury możemy więc określić linię, która zmieniła swój stan. Skoro potrafimy już obsługiwać linie I/O, pozostało nam jeszcze poznanie sposobu wysyłania oraz obsługi wiadomości tekstowych za pomocą funkcji API modułu. Podobnie jak poprzednio musimy najpierw zadeklarować chęć obsługi SMS–ów wewnątrz modułu, do czego służy funkcja *adl_smsSubscribe*, która ma następujący prototyp oraz przyjmuje następujące parametry:

```
s8 adl_smsSubscribe(adl_smsHdlr_f SmsHandler, adl_
smsCtrlHdlr_f SmsCtrlHandler, u8 Mode);
```

SmsHandler – Jest wskaźnikiem do funkcji obsługi zdarzenia, które jest wywoływane w momencie odebrania wiadomości tekstowej.

SmsCtrlHandler – Jest wskaźnikiem do funkcji obsługi zdarzenia informującego o statusie wysyłania wiadomości tekstowej.

Mode – Tryb obsługi SMS–ów. Interesować nas będzie obsługa wiadomości tylko w trybie tekstowym, dlatego pole to powinno zawsze przyjmować wartość ADL_SMS_MODE_TXT.

Jeżeli realizacja funkcji zakończyła się powodzeniem, wówczas zwracana jest nieujemna wartość będąca uchwytem, który powinien być wykorzystywany przez inne funkcje obsługi wiadomości SMS.

Funkcja obsługi zdarzenia od odebrania wiadomości tekstowej powinna być zdefiniowana następująco:

```
bool sms_msg_callback(ascii *tel,ascii *time,ascii
*text);
```

Funkcja ta jest wywoływana w momencie, gdy zostanie odebrana wiadomość tekstowa. Jeżeli nie chcemy odbierać wartości tekstowych, tak jak w przypadku naszego programu, który tylko wysyła SMS–y, funkcja ta powinna być pustą funkcją zwracającą wartość TRUE, co oznacza, że dalsza obróbka odebranej wiadomości powinna być wykonana przez system operacyjny modułu. Z kolei funkcja obsługi zdarzeń kontrolnych SMS powinna być zdefiniowana następująco:

```
void sms_ctrl_callback(u8 event,u16 nb);
```

Funkcja ta jest wywoływana w momencie wystąpienia zdarzenia informującego o wysłaniu, bądź nie, wiadomości tekstowej. Parametr event może przyjąć następujące wartości: ADL_SMS_EVENT_SENDING_OK w przypadku, gdy udało się wysłać wiadomość tekstową (wówczas parametr nb nie jest istotny) lub ADL_SMS_EVENT_SENDING_ERROR w przypadku, gdy nie udało się wysłać wiadomości (wówczas parametr nb określa numer błędu.

Do rozpoczęcia wysyłania wiadomości SMS służy funkcja:

```
s8 adl_smsSend(u8 Handle, ascii *SmsTel, ascii
*SmsText, u8 Mode);.
```

Jako pierwszy parametr przyjmuje ona uchwyt, który zwraca funkcja *adl_smsSubscribe*, drugi parametr zawiera numer telefonu w postaci tekstowej, trzeci parametr zawiera tekst jaki chcemy wysłać, natomiast ostatni parametr zawiera tryb wysyłania wiadomości tekstowej, który w naszym przypadku powinien zawsze przyjmować wartość ADL_SMS_MODE_TEXT. Funkcja ta zwraca wartość OK w przypadku, gdy udało się rozpocząć proces wysyłania wiadomości. Zwrócenie wartości OK wcale nie oznacza, że wiadomość została wysłana poprawnie. Poprawne wysłanie wiadomości potwierdzane jest dopiero za pomocą zdarzenia ADL_SMS_EVENT_SENDING_OK.

Poznaliśmy już w zasadzie wszystkie funkcje niezbędne do napisania naszej aplikacji. Jej kod przedstawimy w następnym odcinku.

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

R E K L A M A