Technologia GSM w elektronice (4)

Programowanie w Open AT

W poprzednim odcinku naszego cyklu zapoznaliśmy się ze środowiskiem programistycznym oraz dostępnymi dla programisty narzędziami. Stworzyliśmy również pierwsza prostą aplikację. Teraz przyszła kolej na dalsze poznawanie możliwości Open AT.

Na początek zajmiemy się zagadnieniem obsługi karty SIM. Moduły GSM, podobnie jak telefony komórkowe, do prawidłowej pracy w sieci potrzebują karty SIM. *Open AT* daje programiście możliwość wprowadzenia kodów PIN i PUK związanych z kartą SIM oraz sprawdzenia statusu karty. Stwórzmy zatem nową aplikację, która będzie automatycznie wpisywała PIN oraz wyświetlała status karty. Sposób tworzenia aplikacji został dokładniej omówiony w poprzednim odcinku cyklu (dla przypomnienia z menu *File* wybieramy *New*, a potem *Open AT Project*).

Obsługa karty SIM, tworzenie nowych komend AT

Przykładową aplikację wpisującą kod PIN przedstawiono na **list. 1**.

Działanie aplikacji rozpoczyna się od funkcji *adl_main()*, w której jest wpisywany kod PIN oraz inicjowana karta SIM i zostaje wskazana funkcja, która będzie wywoływana, gdy wystąpi zdarzenie związane z kartą SIM (w przypadku karty bez numeru PIN należy jako drugi argument funkcji *adl_simSubscribe()* podać *NULL*). Inicjalizacja karty wymaga trochę czasu. Widać to po kolejnych wywołaniach funkcji *SimHandler()* z różnymi zdarzeniami.

Jeśli podano poprawny kod PIN, to aplikacja przez interfejs RS232 prześle następujący ciąg komunikatów:

```
OK
ADL_SIM_EVENT_INSERTED
ADL_SIM_EVENT_PIN_OK
ADL_SIM_EVENT_FULL_INIT
```

Tab. 1. Parametry funkcji adl_atCmdSubscribe

Parametr	Przykład komendy
ADL_CMD_TYPE_PARA	AT+cmd=x, y
ADL_CMD_TYPE_TEST	AT+cmd=?
ADL_CMD_TYPE_READ	AT+cmd?
ADL_CMD_TYPE_ACT	AT+cmd

Zwróćmy uwagę, że inicjalizacja karty chwilę trwa i zdarzenie *ADL_SIM_EVENT_FULL_INIT* zostaje wyświetlone dopiero po pewnym czasie. Ze względu na chęć maksymalnego uproszczenia, w aplikacji nie zawarłem obsługi wszystkich możliwych zdarzeń związanych z kartą SIM. Zostały one dokładnie omówione w dokumentacji („Open AT ADL Development Guide", *Help –> Help Contents –> Open AT Embedded Software Suite package version 6.32*). Można je również zobaczyć, zaznaczając jedno ze zdarzeń i naciskając F3. Nazwy zdarzeń są zdefiniowane w pliku nagłówkowym jako struktura wyliczeniowa. W dokumentacji zostały również opisane pozostałe funkcje dotyczące karty SIM pozwalające m.in. podać kod PUK, sprawdzić status karty, liczbę pozostałych prób wprowadzenia kodu PIN lub PUK.

W kolejnym kroku spróbujmy napisać aplikację, która wpisuje do karty PIN oraz tworzy własną komendę AT pozwalającą wysłać SMS na wskazany numer. Tę aplikację potraktujmy jako ćwiczenie i pomińmy fakt, że istnieje gotowa komenda AT, która na to pozwala. Przykład realizacji postawionego wyżej zadania umieszczono na **list. 2**.

Aplikacja najpierw wpisuje kod PIN za pomocą funkcji *adl_simSubscribe()*. Następnie czeka do momentu, aż karta SIM będzie w pełni zainicjalizowana (*ADL_SIM_STATE_FULL_INIT*). Wtedy subskrybuje się do serwisu SMS oraz tworzy nową komendę *AT+WYSLIJ*. Wystąpienie zdarzenia *ADL_SIM_STATE_FULL_INIT* nie oznacza jednak, że moduł zdążył się już zalogować do sieci. Aby nasza aplikacja działała niezawodnie, należałoby sprawdzić stan zalogowania komendą *AT+CREG?* Takie rozwiązanie zostanie przedstawione w następnym odcinku cyklu, tymczasem w tym odcinku będziemy sprawdzać jedynie wystąpienie zdarzenia *ADL_SIM_STATE_FULL_INIT*.

List. 1. Sposób obsługi karty SIM

```
#include "adl_global.h"
const u16 wm_apmCustomStackSize = 1024*3;

ascii * PinCode = "9172"; //tu wpisać właściwy PIN

void SimHandler(u8 Event){

   TRACE (( 1, "Funkcja SimHandler: Event=%d",Event ));
   switch (Event){
     case ADL_SIM_EVENT_REMOVED:  //< SIM removed event
     adl_atSendResponse( ADL_AT_RSP, "\r\n ADL_SIM_EVENT_REMOVED\r\n");
     break;
   case ADL_SIM_EVENT_INSERTED:  //< SIM inserted event
     adl_atSendResponse( ADL_AT_RSP, "\r\n ADL_SIM_EVENT_INSERTED \r\n");
     break;
   case ADL_SIM_EVENT_FULL_INIT:  //< SIM Full Init done event
     adl_atSendResponse( ADL_AT_RSP, "\r\n ADL_SIM_EVENT_FULL_INIT\r\n");
     break;
   case ADL_SIM_EVENT_PIN_ERROR:  //< Wrong PUK input event
     adl_atSendResponse( ADL_AT_RSP, "\r\n ADL_SIM_EVENT_PIN_ERROR\r\n");
     break;
   case ADL_SIM_EVENT_PIN_OK:  //< PIN code OK event
     adl_atSendResponse( ADL_AT_RSP, "\r\n ADL_SIM_EVENT_PIN_OK\r\n");
     break;
   case ADL_SIM_EVENT_PIN_WAIT:  //< PIN wait event
     adl_atSendResponse( ADL_AT_RSP, "\r\n ADL_SIM_EVENT_PIN_WAIT\r\n");;
     break;
}//end switch
}

void adl_main ( adl_InitType_e InitType )
{
   TRACE (( 1, "Funkcja Main" ));
   adl_simSubscribe( SimHandler, PinCode );

}
```

Subskrypcja do lokalnej usługi SMS za pomocą *adl_smsSubscribe()* polega na wskazaniu dwóch funkcji oraz trybu, w jakim chcemy wysyłać SMS-y. Pierwsza ze wskazywanych funkcji będzie wywołana przez system w momencie, gdy nasze urządzenie odbierze SMS. Zmiennymi wejściowymi tej funkcji są ciągi znaków zawierające numer telefonu, od którego otrzymaliśmy wiadomość oraz jej treść. W opisywanej aplikacji są one po sformatowaniu wysyłane przez port szeregowy. Wartość zwracana przez funkcję (w naszym przypadku *ADL_SMS_FILTER_INDICATION_AND_DELETE*) mówi, czy SMS ma zostać zapisany na karcie SIM, a powiadomienie o jego otrzymaniu (*+CMTI*) wysłane przez port szeregowy.

Druga z funkcji, które podajemy jako argument *adl_smsSubscribe()*, to funkcja kontrolna. Jest wywoływana, gdy nasze urządzenie wysłało SMS. Zmienna wejściowa tej funkcji mówi o tym, jakim rezultatem zakończył się proces wysyłania – powodzeniem lub błędem (nie należy mylić z raportem o doręczeniu). Informacje przekazywane przez tę funkcję są takie same, jak zwykle obserwowane na ekranie telefonu komórkowego, a mówiące o tym, czy SMS został wysłany pomyślnie.

List. 2. Sposób tworzenia nowej komendy AT

```
#include „adl_global.h”

const u16 wm_apmCustomStackSize = 1024*3;
ascii * PinCode = „9172”;
s8 sms_handle;

bool SmsHandler ( ascii * SmsTel, ascii * SmsTimeOrLength, ascii * SmsText ){

   ascii tekst[0x50];
   wm_sprintf (tekst,” Numer telefonu: %s \n\r”,SmsTel );
   adl_atSendResponse ( ADL_AT_UNS, tekst );
   wm_sprintf (tekst,” Tresc wiadomosci: %s \n\r”,SmsText );
   adl_atSendResponse ( ADL_AT_UNS, tekst );
   return ADL_SMS_FILTER_INDICATION_AND_DELETE;
}

void SmsCtrlHandler(u8 Event, u16 Nb)
{
   if (Event == ADL_SMS_EVENT_SENDING_ERROR)
      adl_atSendResponse ( ADL_AT_UNS, „\r\n SMS nie zostal wyslany\r\n” );
   else if (Event == ADL_SMS_EVENT_SENDING_OK)
      adl_atSendResponse ( ADL_AT_UNS, „\r\n SMS zostal wyslany pozytywnie\r\n” );
}

void Fun_wyslij(adl_atCmdPreParser_t * param) {
   ascii * NR_tel;
   ascii * SMS_Content;
   if (param->Type == ADL_CMD_TYPE_PARA){  //sprawdź czy wywołanie z parametrami
      NR_tel = ADL_GET_PARAM ( param, 0 );  //pobierz pierwszy parametr
      TRACE (( 3,NR_tel));
      SMS_Content = ADL_GET_PARAM ( param, 1 );  //pobierz drugi parameter
      TRACE (( 3,SMS_Content));
      adl_smsSend( sms_handle, NR_tel, SMS_Content, ADL_SMS_MODE_TEXT );
      adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);  //wyświetl OK
   }
   else {
      adl_atSendResponse ( ADL_AT_UNS, „\r\n Tylko wywolanie z parametrami\r\n” );
      adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_ERROR);
   }
}

void SimHandler(u8 Event){

   if (Event == ADL_SIM_STATE_FULL_INIT){ //sprawdź czy karta w pelni gotowa
      sms_handle = adl_smsSubscribe(SmsHandler, SmsCtrlHandler, ADL_SMS_MODE_TEXT );
      adl_atCmdSubscribe(„AT+WYSLIJ”, Fun_wyslij, ADL_CMD_TYPE_PARA | 0x0022);
   }
}

void adl_main ( adl_InitType_e InitType )
{
   TRACE (( 1, „Embedded Application : Main” ));
   adl_simSubscribe(SimHandler,PinCode);
}
```

Nową komendę AT tworzy funkcja *adl_atCmdSubscribe()*. Jako jej argumenty podaje się:

- ciąg znakowy będący treścią komendy (musi zaczynać się od „AT”),
- funkcję, która zostanie wywołana, jeśli komenda będzie odebrana przez którykolwiek z aktywnych portów szeregowych,
- iloczyn logiczny parametrów określający, w jakich trybach może być wywołana komenda.

Zestawienie parametrów określających możliwe tryby wywołania komendy umieszczono w **tab. 1**.

Podczas tworzenia nowej komendy AT jest możliwe ograniczenie liczby przyjmowanych parametrów. Zilustrują to kolejne dwa przykłady. Deklaracja w postaci *adl_atCmdSubscribe(„AT+WYSLIJ”, Fun_wyslij, ADL_CMD_TYPE_PARA | 0x0022);* spowoduje utworzenie nowej komendy *AT+WYSLIJ* jako komendy z parametrami. Wymagane są minimalnie oraz maksymalnie 2 parametry. Niezgodne wywołanie komendy spowoduje, że system odpowie błędem (komunikat *ERROR*). Jeśli chcemy, aby komenda przyjmowała co najmniej jeden parametr, ale nie więcej niż trzy, to deklaracja musiałaby wyglądać następująco: *adl_atCmdSubscribe(“AT+WYSLIJ”, Fun_wyslij, ADL_CMD_TYPE_PARA | 0x0013);*

List. 3. Obsługa cyfrowych linii GPIO

```
include "adl_global.h"
const u16 wm_apmCustomStackSize = 1024*3;
adl_ioDefs_t Wejscia[2];
adl_ioDefs_t Wyjscie[1];
s32 io_handle_we;
s32 io_handle_wy;
s32 GpioEventHandle;
bool Stan_diody = FALSE;

void GPIO_TELE_handler( s32 gpio_handle, adl_ioEvent_e Event, u32 Size, void * Param )
{
        ascii tekst [30] ={0};
        TRACE (( 1, "Gpio event %d / %d", Event, Size ));
        // Switch on event
        if ( Event == ADL_IO_EVENT_INPUT_CHANGED ) {
                if(((((adl_ioDefs_t*)Param)[0]) & ADL_IO_NUM_MSK) == 19){
                        TRACE (( 1, "GPIO %d new value: %d",
                        (((adl_ioDefs_t *)Param)[0]) & ADL_IO_NUM_MSK,
                        ((((adl_ioDefs_t *)Param)[0] ) & ADL_IO_LEV_MSK ) & ADL_IO_LEV_HIGH ));
                        wm_sprintf(tekst,"\r\n Zmieniono stan GPIO 19 na %d\r\n",
                        adl_ioReadSingle ( io_handle_we, &Wejscia[0]) );
                        adl_atSendResponse ( ADL_AT_UNS, tekst);
                }
                if(((((adl_ioDefs_t*)Param)[0]) & ADL_IO_NUM_MSK) == 23){
                        TRACE (( 1, "GPIO %d new value: %d",
                        (((adl_ioDefs_t *)Param)[0] ) & ADL_IO_NUM_MSK ,
                        ((((adl_ioDefs_t *)Param)[0] ) & ADL_IO_LEV_MSK ) & ADL_IO_LEV_HIGH ));
                        wm_sprintf(tekst,"\r\n Zmieniono stan GPIO 23 na %d\r\n",
                        adl_ioReadSingle ( io_handle_we, &Wejscia[1] ) );
                        adl_atSendResponse ( ADL_AT_UNS,tekst);
                }
        }
}


void GPIO_TimerHandler ( u8 ID )
{
        /* Hello World */
        TRACE (( 1, "Embedded : GPIO_TimerHandler" ));
        if(Stan_diody){
                adl_ioWriteSingle(io_handle_wy, &Wyjscie[0], TRUE);
                Stan_diody = FALSE;
        }
        else{
                adl_ioWriteSingle(io_handle_wy, &Wyjscie[0], FALSE);
                Stan_diody = TRUE;
        }
}


void adl_main ( adl_InitType_e InitType )
{
        TRACE (( 1, "Embedded Application : Main" ));
        Wejscia[0] = ADL_IO_GPIO | 19 | ADL_IO_DIR_IN;   //pin45
        Wejscia[1] = ADL_IO_GPIO | 23 | ADL_IO_DIR_IN;     //pin55
        Wyjscie[0] = ADL_IO_GPIO | 24 | ADL_IO_DIR_OUT |ADL_IO_LEV_LOW; //pin58
        GpioEventHandle = adl_ioEventSubscribe ( GPIO_TELE_handler );
        io_handle_we = adl_ioSubscribe (2, Wejscia, ADL_TMR_TYPE_100MS, 1, GpioEventHandle );
        io_handle_wy = adl_ioSubscribe ( 1, Wyjscie, 0, 0, 0 );
        adl_tmrSubscribe ( TRUE, 20, ADL_TMR_TYPE_100MS, GPIO_TimerHandler );
}
```

Tab. 2. Parametry elektryczne cyfrowych linii I/O i wewnętrznego zasilacza

Parametr		Typ I/O	Minimum	Typowo	Maksimum	Uwagi
Wewnętrzne zasilanie 2,8 V		VCC_2V8	2,74 V	2,8 V	2,86 V	
Doprowadzenie I/O	VIL	CMOS	–0,5 V		0,84 V	
	VIH	CMOS	1,96 V		3,2 V	IOL=–4 mA
	VOL	CMOS			0,4 V	IOL=–4 mA
	VOH	CMOS	2,4 V			IOH=4 mA
	IOH	CMOS			4 mA	
	IOL	CMOS			-4 mA	

Obsługa cyfrowych linii GPIO

Kolejnym zagadnieniem jest obsługa cyfrowych linii wejścia/wyjścia – GPIO. W przedstawionym niżej przykładzie wykorzystano trzy linie cyfrowe. Dwie z nich – GPIO19 i GPIO23 – pracują jako wejścia cyfrowe, natomiast GPIO24 jako wyjście. Linie pracują w logice CMOS z napięciem logicznej „1" wynoszącym 2,8 V. Ich parametry elektryczne zamieszczono w **tab. 2**.

Przestrzegając parametrów zamieszczonych w tab. 1, do linii wyjścia GPIO24 podłączymy diodę z odpowiednio dobranym rezystorem, a do linii wejść cyfrowych GPIO19 i GPIO23 prosty układ rezystorowy z przełącznikiem, pozwalający na podawanie naprzemiennie stanu niskiego lub wysokiego.

Przykładową aplikację demonstrującą działanie GPIO pokazano na **list. 3**. Działanie aplikacji rozpoczyna się od zdefiniowania linii GPIO oraz nadania im odpowiednich funkcji (dwa wejścia i jedno wyjście). Dla linii wejściowych definiowana jest funkcja zdarzeniowa, która będzie wywoływana przez system, gdy na którejś z linii nastąpi zmiana. Podczas subskrypcji linii wejściowych za pomocą funkcji *adl_ioSubscribe()* okres próbkowania zostaje ustawiony na 100 ms (najkrótszy okres próbkowania to 18,5 ms). Na końcu funkcji *adl_main()* jest wywołany cykliczny timer o okresie 2 s, wywołujący funkcję *GPIO_TimerHandler()*. W funkcji tej stan linii GPIO24 zostaje zmieniony na przeciwny do przedniego. W ten sposób na tym wyjściu jest generowany przebieg prostokątny o okresie 4 s.

Funkcja zdarzeniowa *GPIO_TimerHandler()* będzie wywoływana przez system zawsze, gdy nastąpi zmiana na zadeklarowanych liniach GPIO. Wewnątrz funkcji następuje sprawdzenie, stan której linii został zmieniony (możliwa jest sytuacja, że obie linie zmienią swój stan). Stan linii po zmianie można sprawdzić na dwa sposoby: wykorzystując zmienną wejściową *Param* lub odczytując stan linii funkcją *adl_ioReadSingle()*.

List. 4. Aplikacja łącząca obsługę GPIO i obsługę karty SIM

```
#include „adl_global.h”
const u16 wm_apmCustomStackSize = 1024*3;
adl_ioDefs_t Wyjscie[1];
ascii * PinCode = "9172";
s32 io_handle_wy;
adl_ioDefs_t Wyjscie[1];
s8 sms_handle;

bool SmsHandler ( ascii * SmsTel, ascii * SmsTimeOrLength, ascii * SmsText ){
        TRACE((1,"SMS Received"));
        if (wm_strcmp(SmsText,"HIGH")==0){
                adl_ioWriteSingle(io_handle_wy, &Wyjscie[0], TRUE);
                return ADL_SMS_FILTER_INDICATION_AND_DELETE;
        }
        if (wm_strcmp(SmsText,"LOW")==0){
                adl_ioWriteSingle(io_handle_wy, &Wyjscie[0], FALSE);
                return ADL_SMS_FILTER_INDICATION_AND_DELETE;
        }
        adl_smsSend(sms_handle, SmsTel, "Niewlasciwa komenda", ADL_SMS_MODE_TEXT );
        return ADL_SMS_FILTER_INDICATION_AND_DELETE;
}


void SmsCtrlHandler(u8 Event, u16 Nb)
{
        if (Event == ADL_SMS_EVENT_SENDING_ERROR)
                adl_atSendResponse ( ADL_AT_UNS, "\r\n SMS nie zostal wyslany\r\n" );
        else if (Event == ADL_SMS_EVENT_SENDING_OK)
                adl_atSendResponse ( ADL_AT_UNS, "\r\n SMS zostal wyslany poprawnie\r\n" );
}


void SimHandler(u8 Event){
        TRACE((1,"Event = %d", Event));
        if (Event == ADL_SIM_STATE_FULL_INIT)  //sprawdź czy karta gotowa
                sms_handle = adl_smsSubscribe(SmsHandler, SmsCtrlHandler, ADL_SMS_MODE_TEXT );
}


void adl_main ( adl_InitType_e InitType )
{
        TRACE (( 1, "Embedded Application : Main" ));
        Wyjscie[0] = ADL_IO_GPIO | 24 | ADL_IO_DIR_OUT |ADL_IO_LEV_LOW; //pin58
        io_handle_wy = adl_ioSubscribe ( 1, Wyjscie, 0, 0, 0 );
        adl_simSubscribe(SimHandler,PinCode);

}
```

Podsumowanie

Opisane serwisy oferują znacznie więcej możliwości, niż zostało to opisane, o czym można się łatwo przekonać, czytając dokumentację biblioteki ADL – „*Open AT ADL Development Guide*”. Podsumowując ten odcinek, warto przedstawić aplikację łączącą omawiane usługi: SIM, SMS i GPIO. Jej przykład zamieszczono na **list. 4.**

Aplikacja najpierw wpisuje PIN do karty SIM, a następnie czeka na pełną inicjalizację. Po odebraniu SMS-a z komendą LOW lub HIGH aplikacja ustawi odpowiednio stan wyjścia GPIO24. W przypadku, gdy komenda różni się od dwóch zadeklarowanych, zostaje odesłany SMS o treści „*Niewlasciwa komenda*”.

Więcej informacji na temat produktów Sierra Wireless można znaleźć na stronach producenta: *www.sierrawireless.com* lub kontaktując się z firmą ACTE Sp. z o.o., która jest oficjalnym dystrybutorem opisywanych produktów oraz zapewnia pełne wsparcie techniczne.

Adrian Chrzanowski
Acte Sp. z o.o.

R E K L A M A