Technologia GSM w elektronice (5)

Open AT – komendy AT, pamięć Flash

W poprzednim odcinku zajmowaliśmy się między innymi tworzeniem własnych komend AT. W tym pokażemy, jak stosować do swoich celów komendy AT oferowane przez system operacyjny. Pokażemy również, jak wykorzystać pamięć nieulotną do trwałego przechowywania danych.

Na jakość działania i niezawodność projektu M2M duży wpływ ma to, czy na etapie projektowania będziemy w stanie przewidzieć wszystkie okoliczności, które mogą wystąpić podczas pracy urządzenia. W poprzednim odcinku, gdy opisywaliśmy serwis SIM oraz SMS, dla uproszczenia przyjąłem, że *ADL_SIM_EVENT_FULL_INIT* oznacza, że modem jest zalogowany i można już wysyłać SMS. Tak jednak będzie tylko wtedy, gdy modem znajduje się w miejscu, w którym jest dobry sygnał GSM, a sieć działa prawidłowo i nie jest przeciążona. Aby jednak bez względu na okoliczności być pewnym, że modem jest zalogowany, posłużymy się komendą AT+CREG. Komenda ta, wydana ze znakiem zapytania, zwraca status zalogowania się aparatu do sieci. W aplikacji pokazanej na **listingu 1** komenda ta jest wydawana cyklicznie, aż do otrzymania odpowiedzi „1” (zalogowany) lub „5” (zalogowany w roamingu).

Zaprezentowana aplikacja w większości przedstawia to, czego już nauczyliśmy się z poprzednich odcinków kursu z tym, że tu SMS zostanie wysłany tylko wtedy, kiedy modem faktycznie zaloguje się do sieci. Dla przykładu, jeśli odłączymy antenę GSM, to mimo iż zajdzie zdarzenie *ADL_SIM_EVENT_FULL_INIT*, to jednak SMS nie zostanie wysłany. Wszystko to dzięki kontroli odpowiedzi na komendę *AT+CREG?*. Sprawdźmy jak wygląda odpowiedź modemu, jeśli komendę *AT+CREG?* wydamy do modemu z terminala (odpowiedź modemu napisano pogrubioną czcionką):

```
AT+CREG?
+CREG: 0,1

OK
```

Mamy dwie odpowiedzi: jedna to „*+CREG: 0,1*”, natomiast druga to „*OK*”. W naszej aplikacji wydając komendę *AT+CREG?* za pomocą funkcji *adl_atCmdCreate()* podajemy funkcję, która będzie otrzymywała odpowiedź na wskazaną komendę AT, a także filtr odpowiedzi, które chcemy przechwytywać. W na-

Listing 1. Sprawdzanie stanu zalogowania się modemu do sieci GSM

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

#define GPRS_PINCODE  "9172"
s8 sms_handle;

static void poll_creg( u8 Id );


void Wyslij_SMS(){

        adl_smsSend( sms_handle, "+48xxxxxxxxx", "TEST SMS", ADL_SMS_MODE_
TEXT );
}


static bool poll_creg_callback(adl_atResponse_t *Rsp) {

    ascii regStateString[3];
    s32 regStateInt;

    TRACE (( 1, "poll_creg_callback" ));

    wm_strGetParameterString(regStateString, Rsp->StrData, 2);
    regStateInt = wm_atoi(regStateString);

    if ( 1 == regStateInt || 5 ==regStateInt) {
        TRACE (( 1, „poll_creg_callback - zalogowany w sieci" ));
        Wyslij_SMS();
    } else {
      /* Nie gotowy - sprawdz ponownie za 1s */
      adl_tmrSubscribe( FALSE, 10, ADL_TMR_TYPE_100MS,
                        poll_creg);
    }
    return FALSE;
}

static void poll_creg( u8 Id ) {
  adl_atCmdCreate( "AT+CREG?", FALSE, poll_creg_callback, ADL_STR_CREG, NULL);
}

static void SIM_Handler( u8 event) {
  TRACE (( 1, "SIM_Handler" ));
  if(  event == ADL_SIM_EVENT_FULL_INIT) {
          TRACE (( 1, "SIM Full Init" ));

          poll_creg(0);
  }
}

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

        return TRUE;
}

void SMS_Control ( u8 Event, u16 Nb ){
}

void adl_main ( adl_InitType_e InitType )
{
    TRACE (( 1, "Embedded Application : Main" ));
    adl_simSubscribe( SIM_Handler, GPRS_PINCODE);
        sms_handle = adl_smsSubscribe(SMS_handler, SMS_Control, ADL_SMS_MODE_
TEXT );
        TRACE((1,"SMS handle = %d", sms_handle));

}
```

Listing 2. Przykład konfigurowania portu szeregowego

```
#include "adl_global.h"

const u16 wm_apmCustomStackSize = 1024*3;

void Wylacz ( u8 ID, void * Context){
        TRACE((1,"Wylacz UART2"));
        adl_atCmdCreate ( "AT+WMFM=0,0,2" ,FALSE , NULL, NULL);
}

bool Conf_Handler(adl_atResponse_t * strc){

     TRACE (( 1, "Conf_Handler: response = %d",strc->RspID ));
    if(strc->RspID==ADL_STR_OK) {
        adl_atSendResponse( ADL_AT_PORT_TYPE(ADL_AT_UART2,ADL_AT_RSP),"Hello
World UART2\r\n" );
        adl_tmrSubscribe(FALSE,10,ADL_TMR_TYPE_100MS,Wylacz);

    }
  return TRUE;
}

bool Open_Handler(adl_atResponse_t * strc){

     TRACE (( 1, "Open_Handler: response = %d",strc->RspID ));
    if(strc->RspID==ADL_STR_OK || strc->RspID==ADL_STR_CME_ERROR) {
        adl_atCmdCreate ( "AT+IPR=9600;+IFC=0,0" ,ADL_AT_PORT_TYPE(ADL_AT_
UART2, FALSE), Conf_Handler , "*" , NULL);
    }

  return TRUE;
}

void adl_main ( adl_InitType_e InitType )
{
    TRACE (( 1, "Embedded Application : Main" ));
    adl_atCmdCreate ( "AT+WMFM=0,1,2" ,FALSE , Open_Handler , "*" , NULL);
}
```

Listing 3. Przykład użycia pamięci flash obiektów do przechowywania jednorazowo wprowadzonego kodu PIN

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

/*****************************************************************/
/*  Local variables                                              */
/*****************************************************************/
static const ascii * hand_name = "kod pin w pamieci flash";

/*****************************************************************/
/*  Local functions                                              */
/*****************************************************************/
void SimHandler ( u8 Event )
{
    TRACE (( 1, "Embedded : SimHandler, Event=%d",Event ));

    switch (Event){

        case ADL_SIM_EVENT_FULL_INIT:
            adl_atSendResponse ( ADL_AT_UNS, "karta SIM zainicjalizowana\n\r"
);

            break;
        case ADL_SIM_EVENT_PIN_OK:
            adl_atSendResponse ( ADL_AT_UNS, "SIM PIN ok\n\r" );
            break;

    }//end switch

}//end simHandler

void Fun_pin(adl_atCmdPreParser_t * param) {

ascii * pin;

    TRACE((1,"W Fun_pin"));

    if (param->Type == ADL_CMD_TYPE_PARA){

        pin = ADL_GET_PARAM ( param, 0 );  //pobierz parametr komendy PIN
        TRACE (( 3,pin));
        TRACE((3,"Strlen of pin = %d",strlen(pin) ));
    }

    if(pin!=NULL){
        adl_flhWrite( hand_name, 0, wm_strlen(pin)+1, (u8 *) pin ); //zapisz
pin do p.flash
        adl_simSubscribe (SimHandler, pin ); //wpisz PIN do karty
     }
     adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
}

void Fun_del(adl_atCmdPreParser_t * param){

        if (adl_flhExist( hand_name, 0 )>0){
                adl_flhErase( hand_name, 0 );
        }
        adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
}


void adl_main ( adl_InitType_e InitType )
```

szym przypadku jest to *ADL_STR_CREG*, czyli jak w definicji – string w postaci: „*+CREG:*”. Jako filtr możemy podać kilka ciągów znakowych oddzielonych przecinkiem i zakończonych wartością NULL lub jeśli chcemy przechwytywać wszystkie możliwe odpowiedzi, to jako argument podajemy „*”. Za pomocą funkcji *wm_strGetParameterString()* pobieramy drugi parametr odpowiedzi i sprawdzamy czy jest równy wartości „1” lub „5”. Jeśli tak, to wykonywana jest funkcja *Wyslij_SMS*, w przeciwnym razie uruchamiany jest jednorazowy timer, który po sekundzie ponownie wywoła funkcję pytającą o stan zalogowania. Funkcja *static void poll_creg(u8 Id)* może być bez argumentów, ale wykorzystywana jest również jako funkcja – handler dla timera, a wtedy już musi być w określonej postaci. Jeśli chcielibyśmy zobaczyć przechwyconą odpowiedź w całości, to najlepiej wewnątrz funkcji *poll_creg_callback()* dodać linijkę:

```
TRACE((1,Rsp->StrData));
```

W przedstawiony powyżej sposób można również odczytywać odbierany poziom sygnału GSM. Należałoby w takim przypadku skorzystać z komendy *AT+CSQ*.

Wydawanie komend AT wewnątrz aplikacji jest bardzo przydatna, możliwości nie tylko do celów diagnostycznych, ale również konfiguracyjnych. Na **listingu 2** umieszczono przykład konfigurujący port szeregowy. Aplikacja włącza port UART2 komendą *AT+WMFM=0,1,2* i czeka na odpowiedź. Wyjątkowo w tym przypadku, oprócz odpowiedzi *OK* za poprawną przyjmowana jest również odpowiedź *CME_ERROR*. Zastosowano takie uproszczenie, aby pominąć sprawdzanie czy UART2 jest włączony przed wydaniem komendy. Próba ponownego włączenia już włączonego portu generuje właśnie odpowiedź ERROR. Jeśli wiemy, że tak właśnie jest, to takie rozwiązanie jest do zaakceptowania. Po wydaniu komendy włączającej port, aplikacja ustawia następujące parametry transmisji: prędkość 9600 bps oraz wyłączona kontrola przepływu. Wydanie komendy *AT+IPR=9600;+IFC=0,0* jest równoznaczne z wydaniem dwóch komend

```
AT+IPR=9600
AT+IFC=0,0
```

W odpowiedzi na komendę sklejoną otrzymamy *OK* tylko wtedy, jeśli obie komendy zwrócą odpowiedzi *OK*.

Po ustawieniu żądanych parametrów, przez UART2 zostaje wysłany komunikat, a następnie port zostaje wyłączony. Należy zwrócić uwagę na zastosowane makro *ADL_AT_PORT_TYPE()*. W przypadku funkcji, która włącza i wyłącza UART2, po komendzie AT pojawia się wartość *FALSE*. Oznacza ona, że odpowiedzi które pomijamy przy przechwytywaniu również nie pojawią się w oknie terminala. Komenda AT jest w tym przypadku wydawana w sposób „anonimowy”. W przypadku ustawiania parametrów

portu musimy jawnie wskazać, którego portu te ustawienia mają dotyczyć. W tym celu stosujemy właśnie makro *ADL_AT_PORT_TYPE()*. Pierwszym argumentem jest port, a drugim flaga o znaczeniu analogicznym, jak w przypadku wywołania „anonimowego". To samo makro zostało również wykorzystane w przypadku funkcji wysyłającej tekst *adl_atSendResponse()*.

Więcej szczegółów dotyczących komend AT wykorzystanych w przykładach znajdziemy dokumentacji (*Help –> Help Contents –> Open AT Firmware Documentation –> Firmware version R7.43*)

W modułach Air Prime Q26 Pamięć Flash podzielono na następujące obszary:
- pamięć Flash obiektów,
- pamięć Application and Data (A&D),
- pamięć kodu aplikacji.

Pamięć Flash obiektów jest to pamięć o rozmiarze 384 kB (wielkość ta może być zwiększona kosztem pamięci A&D do 1728 kB za pomocą aplikacji DWLWin) pozwalająca na zapisanie ponad 5000 obiektów dowolnego typu, np. tablice, struktury, zmienne. Maksymalny rozmiar pojedynczego obiektu to 30 kB. W obszarze tym, w sposób niewidoczny dla programisty, pracuje mechanizm zwany *Garbage Collector*. Powoduje on równomierne wykorzystanie całego obszaru pamięci w taki sposób, że gdy zmieniamy wartość pewnego obiektu, to nie jest on zapisywany w tym samym obszarze pamięci. Tak dzieje się do momentu, aż obszar będzie całkowicie wykorzystany. Wtedy zadziała *Garbage Collector* usuwając zwolnione komórki i wykonując defragmentację pamięci. Mechanizm ten przedłuża czas życia pamięci Flash.

Obszar A&D jest również przeznaczony na dane użytkownika, ale oprócz tego może być wykorzystany do przechowania plików firmware'u lub aplikacji w przypadku, gdy korzysta się ze zdalnej aktualizacji. Obszar ten jest dzielony z obszarem kodu aplikacji, a jego rozmiar wynosi od 0 do 4864 kB. Zapisywanie danych w tym obszarze ogranicza jedynie dostępna wielkość pamięci. Obszar kodu aplikacji to miejsce, które jest zajęte przez obecnie działającą aplikację. Rozmiar tego obszaru to od 256 do 5120 kB. Proporcje pamięci przydzielone dla dwóch ostatnich obszarów można zmieniać komendą *AT+WOPEN=6*, np.:

```
AT+WOPEN=6
+WOPEN: 6,2560,2560
```

Odpowiedź zawiera ilość pamięci odpowiednio dla pamięci A&D i kodu aplikacji. Jeśli wydamy komendę: *AT+WOPEN=6,4864*, to otrzymamy odpowiedź: *+WOPEN: 6,4864,256*. Należy pamiętać, że zmieniając proporcje pamięci powodujemy formatowanie całego obszaru pamięci A&D oraz kodu aplikacji. Więcej informacji można znaleźć w dokumentacji komend AT oraz

Listing 3. c.d.

```
{
    ascii * pin;
    TRACE (( 1, "Embedded Application : Main" ));

    if (adl_flhExist( hand_name, 0 )>0){   //czy PIN zostal zapisany w p.flash
         pin = (ascii *)adl_memGet(8);
         wm_memset(pin, 0 ,8);
         adl_flhRead( hand_name, 0,8,(ascii *) pin);
         adl_simSubscribe (SimHandler, pin ); //WPISZ pin DO KARTY
        adl_memRelease (pin);
    }
    else{
        adl_flhSubscribe(hand_name, 1);
    }

    adl_atCmdSubscribe("AT+PIN", Fun_pin, ADL_CMD_TYPE_PARA | 0x0011);
    adl_atCmdSubscribe("AT+DEL", Fun_del, ADL_CMD_TYPE_ACT );
}
```

Listing 4. Przykład użycia pamięci *Application & Data*

```
#include "adl_global.h"

/*************************************************************/
/*  Mandatory variables                                      */
/*-----------------------------------------------------------*/
/*  wm_apmCustomStackSize                                    */
/*-----------------------------------------------------------*/
/*************************************************************/
const u16 wm_apmCustomStackSize = 1024*3;
/*************************************************************/
/*  Local variables                                          */
/*************************************************************/
ascii response[200]={0};
s32 cell_handle=0;
s32 event_handle=0;
/*************************************************************/
/*  Local functions                                          */
/*************************************************************/
void Status(){
        adl_adState_t State;
            adl_adGetState ( &State );
            wm_sprintf(response,"Free memory size %d\r\n Deleted memory size\
t%d\r\n"
                        "Total memory size\t%d\r\nNumber of deleted items\
t%d\r\n"
                        "Number of allocated items\t%d\r\n",
                        State.freemem,State.deletedmem,State.totalmem,State.
numdeleted,
                        State.numobjects);
            adl_atSendResponse(ADL_AT_RSP, response);
            adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
}

void Fun_sub(adl_atCmdPreParser_t * param) {
        TRACE((1,"Inside Fun-sub"));

        ascii cell_s[12];
        s32 cell_size;

        wm_strcpy(cell_s, ADL_GET_PARAM ( param, 0 ));
    cell_size = wm_atoi(cell_s);
    TRACE((1,"size = %d",cell_size));

    cell_handle = adl_adSubscribe ( 1, cell_size );
    TRACE((1,"cell_handle = %d",cell_handle));
    if(cell_handle>=0){
        Status();
    }
    else{
    switch (cell_handle){
        case ADL_RET_ERR_ALREADY_SUBSCRIBED:
                adl_atSendResponse(ADL_AT_RSP,"\r\nCell already subscribed\
r\n");
                break;
        case ADL_AD_RET_ERR_OVERFLOW:
                adl_atSendResponse(ADL_AT_RSP,"\r\nOverflow\r\n");
                break;
        case ADL_RET_ERR_BAD_STATE:
                adl_atSendResponse(ADL_AT_RSP,"\r\nBad State\r\n");
                break;
     }
    }
}
void Fun_del(adl_atCmdPreParser_t * param){

        s32 resp = adl_adDelete ( cell_handle );
        TRACE((1,"Del response= %d",resp));
        adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
        Status();

}
void Fun_wri(adl_atCmdPreParser_t * param){

        TRACE (( 1, "Fun_wri: text length %d",param->StrLength ));

ascii * tekst;

        tekst = ADL_GET_PARAM ( param, 0 );
        //tekst[param->StrLength]=0;
        TRACE((1,tekst));
```

Listing 4. c.d.

```
        if(adl_adWrite ( cell_handle, wm_strlen(tekst), (void *) tekst )>=0){
                adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
        }
        else
                adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_ERROR);
}

void Fun_rea(adl_atCmdPreParser_t * param){

        adl_adInfo_t Dane;

        if(adl_adInfo ( cell_handle, &Dane )== OK){
                wm_sprintf(response,"Rozmiar\t%d\r\nPozostalo\t%d\r\n",Dane.
size,Dane.remaining);
                adl_atSendResponse(ADL_AT_RSP,response);
                wm_memcpy(response,Dane.data,Dane.size-Dane.remaining);
                response[Dane.size-Dane.remaining]=0;
                adl_atSendResponse(ADL_AT_RSP,response);
                TRACE((1,Dane.data));
                adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
        }
        else
                adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_ERROR);
}
void Fun_rec(adl_atCmdPreParser_t * param){

        s32 resp =  adl_adRecompact ( event_handle );
        TRACE((1,"Format response= %d",resp));
        adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
}

void Fun_for(adl_atCmdPreParser_t * param){

        s32 resp = adl_adFormat ( event_handle );
        TRACE((1,"Format response= %d",resp));
        adl_atSendStdResponse(ADL_AT_RSP,ADL_STR_OK);
}

void event_Handler ( adl_adEvent_e Event, u32 Progress ){

        TRACE((1,"event_Handler: Event %d progress %d",Event,Progress));
}

void adl_main ( adl_InitType_e InitType )
{
    TRACE (( 1, "Embedded Application : Main" ));
    //Status();
    adl_atCmdSubscribe("AT+SUB", Fun_sub, ADL_CMD_TYPE_PARA | 0x0011);
    adl_atCmdSubscribe("AT+DEL", Fun_del, ADL_CMD_TYPE_ACT );
    adl_atCmdSubscribe("AT+WRITE", Fun_wri, ADL_CMD_TYPE_PARA | 0x0011);
    adl_atCmdSubscribe("AT+READ", Fun_rea, ADL_CMD_TYPE_ACT );
    adl_atCmdSubscribe("AT+FORMAT", Fun_for, ADL_CMD_TYPE_ACT );
    adl_atCmdSubscribe("AT+RECOMPACT", Fun_rec, ADL_CMD_TYPE_ACT );
    event_handle = adl_adEventSubscribe (&event_Handler);
}
```

w dokumentacji ADL w rozdziale „Memory resources".

Na **listingu 3** umieszczono przykład użycia pamięci flash obiektów do przechowywania jednorazowo wprowadzonego kodu PIN. Aplikacja wprowadza dwie komendy AT – jedną do wprowadzenia PIN-u i drugą do jego usunięcia z pamięci. Wprowadzając PIN do karty za pomocą komendy AT+PIN="xxxx" powodujemy jego jednoczesne zapisanie w pamięci. Jeśli jest to pierwsze uruchomienie lub obiekt został wcześniej usunięty, to aplikacja subskrybuje się do pamięci flash, informując system operacyjny, że będzie wykorzystywała tylko jeden obiekt. Jeśli PIN został już raz wpisany za pomocą komendy AT+PIN, to po każdym resecie lub zaniku zasilania aplikacja sprawdza czy istnieje obiekt, w którym powinien być zapisany PIN, a następnie pobiera go z pamięci za pomocą funkcji *adl_flhRead()* i wpisuje do karty za pomocą wcześniej poznanej funkcji *adl_simSubscribe()*. Proszę zwrócić uwagę, że obiekty przy odczycie i zapisie adresowane są od zera. Istotne jest również, że nie musimy znać wielkości obiektu w momencie subskrypcji pamięci flash, dopiero w momencie jego zapisu do pamięci. W aplikacji zostało również pokazane dynamiczne przydzielanie i zwalnianie pamięci RAM. Całą pamięć obiektów można skasować za pomocą komendy *AT+WOPEN=3* (najpierw jednak AT musi być zatrzymana aplikacja *Open*). Obiekty nie są natomiast kasowane poprzez wgranie nowej aplikacji.

Na **listingu 4** umieszczono przykład użycia pamięci *Application&Data*. Aplikacja tworzy kilka komend AT, dzięki czemu będzie można wygodnie obserwować procesy związane z funkcjonowaniem pamięci.

Na początku należy wydać komendę *AT+SUB="rozmiar"* powodując zadeklarowanie w pamięci komórki o zadeklarowanym rozmiarze wyrażonym w bajtach. Następnie można dokonać zapisu za pomocą komendy *AT+WRITE="dane do zapisania"*. Pamięć A&D różni się też tym od pamięci obiektów, że zapis jest dokonywany sekwencyjnie, a więc kolejne użycie komendy *AT+WRITE* spowoduje dodanie danych po tych, które były zapisane wcześniej. Zapisane wcześniej dane możemy odczytać za pomocą funkcji *AT+READ*. Funkcja ta podaje również rozmiar odczytywanej celi oraz ilość wolnej pamięci w tej konkretnej komórce. Na końcu odczytywanego wektora danych jest dodawane 0, aby mógł być on wyświetlony za pomocą funkcji *adl_atSendResponse()*. W postaci komend AT zostały również zaimplementowane funkcje formatowania oraz kompaktowania pamięci. Rezultaty ich działania są wyświetlane w oknie TRACE przez wcześniej zadeklarowaną funkcję *event_Handler()*.

Do trwałego przechowywania danych można w niektórych przypadkach również użyć obszaru pamięci RAM, który nie jest zerowany po miękkim lub twardym resecie. W celu zapoznania się z tym sposobem przechowywania danych polecam stworzenie i uruchomienie przykładu „*Persistent RAM*", dostępnego wśród aplikacji przykładowych, dostarczanych wraz z IDE.

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.