

Technologia GSM w elektronice (14)

Obsługa modułu GPS za pomocą Open AT

Poprzednio zaprezentowano rozwiązanie z modułem GPS zaproponowane przez Sierrę Wireless. Wówczas podano ogólny opis i zagadnienia związane ze stroną sprzętową. W tym odcinku opiszemy jak obsługiwać moduł XM0110 we własnej aplikacji Open AT.

Poprzedni odcinek naszego kursu był trochę odejściem od wyznaczonego sobie celu, którym jest skupienie się na zagadnieniach związanych z programowaniem modułów GSM Sierra Wireless. Był on jednak swojego rodzaju wstępem, ponieważ trudno byłoby poruszać zagadnienia dotyczące samego programowania, nie opisując uprzednio sprzętu, jego możliwości i parametrów. Podczas testów będziemy używali opisany w poprzednim odcinku zestaw startowy z modułem SL6087 oraz XM0110, w którym do połączenia modułu programowalnego z odbiornikiem GPS wykorzystano interfejs UART2. Oczywiście, nic nie stoi na przeszkodzie, aby testować jakieś własne konstrukcje.

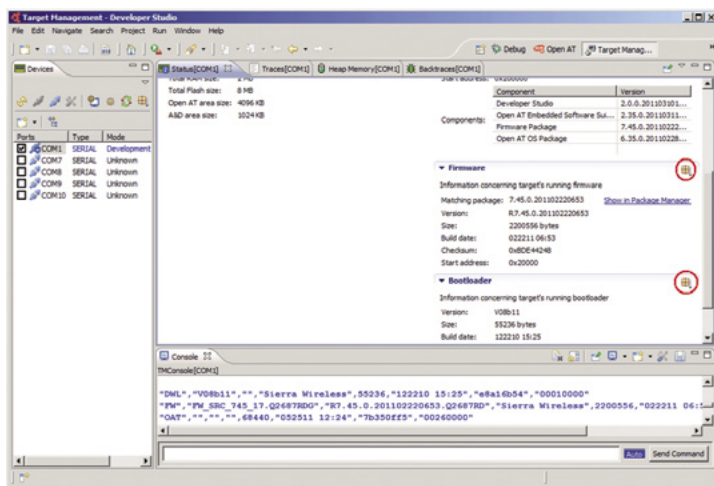
Instalacja Plug-inu oraz aktualizacja firmware

Zanim jednak zaczniemy tworzenie programu obsługującego moduł XM0110 należy upewnić się, czy środowisko Developer

Studio ma zainstalowany pakiet SDK2.35, ponieważ dopiero ta wersja ma *Location Plug-in Package* niezbędny do obsługi modułu XM0110. Jeśli ktoś posiadał starszą wersję środowiska, to powinna ona automatycznie zaktualizować się. Jeśli tak się nie stało, to w menu głównym należy wybrać *Help* a następnie *Check for Updates*.

Jeśli już mamy najnowsze biblioteki, należy zaktualizować firmware w module GSM. Można to zrobić z poziomu *Developer Studio*. W tym celu należy przejść do zakładki *Target Management*, następnie połączyć się z modułem wybierając odpowiedni port COM. Teraz otwieramy zakładkę *Status*. Powinniśmy zobaczyć ekran jak na **rysunku 1**.

Zaznaczono na nim przyciski służące do aktualizacji firmware w module. W pierwszym kroku aktualizujemy bootloader, a następnie firmware, w obu przypadkach wybierając opcję R7.45. Po udanej aktualizacji



Rysunek 1. Widok otwartej zakładki *Status* z zaznaczonymi przyciskami aktualizacji firmware



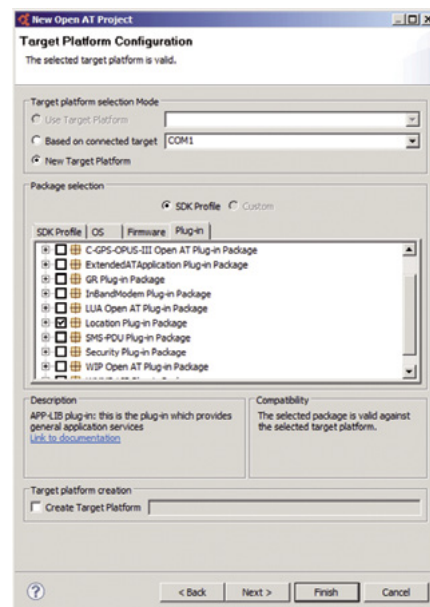
Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 12040, pass: 15735862
• poprzednie części kursu

możemy sprawdzić wersję firmware wydając komendę AT+I3.

Przykładowa aplikacja obsługująca moduł Sierra Wireless XM0110

Po wykonaniu aktualizacji możemy przejść do tworzenia własnej aplikacji obsługującej moduł XM0110. Podczas jej generowania za pomocą kreatora zaznaczamy *Location Plug-in Package* (**rysunek 2**).

Aplikacja zamieszczona na **listingu 1** to uproszczona wersja *SimpleSample* tj. przykładu, który jest dostępny razem ze środowiskiem. W porównaniu z oryginałem pominąłem fragmenty sprawdzające wartości zwracane przez funkcje oraz wszelkiego rodzaju mechanizmy kontrolne, co znacznie



Rysunek 2. Tworzenie aplikacji z wykorzystaniem *Location Plug-in*

Listing 1. Przykład użycia modułu GSM w Open AT

```

#include "adl_global.h"
#include "gps_serv.h"
#include "math.h"

// GPS library tasks must be declared in the Task table definition
extern void Task_GPS_COM( void );
extern void Task_GPS_CORE( void );
extern void Task_GPS_AIDING_AEE( void );

void gps_simpleStart( void );
void user_task( void );
void adl_task( void );
/* GPS stack sizes */
#define GPS_COM_STACK_SIZE          (7*1024)
#define GPS_CORE_STACK_SIZE        (15*1024)
#define GPS_CORE_AIDING_AEE_STACK_SIZE (7*1024)

const u32 wm_apmIRQLowLevelStackSize = 1024*1;
const u32 wm_apmIRQHighLevelStackSize = 1024*1;

static s32 s32VsHandle;
gps_ioConfigXM0110 t_ioConfigXM0110 = {
    GPS_BUS_UART, // interfejs :UART
    GPS_BUS_NUM 2, // UART2
    22, // ON/OFF line -> GPIO22
    23, // Reset Line -> GPIO23
    GPS_GPIO_NOT_MANAGED, // wartość domyślna
    GPS_GPIO_NOT_MANAGED, // wartość domyślna
    GPS_32K_INTERNAL, //sygnał 32 KHz z modułu
};

const adl_InitTasks_t adl_InitTasks [] =
{
    { adl_task, (4*1024), "ADL_TASK", 6 }, // ID = 0
    { Task_GPS_COM, GPS_COM_STACK_SIZE, "GPS1", 5 }, // ID = 1
    { Task_GPS_CORE, GPS_CORE_STACK_SIZE, "GPS0", 4 }, // ID = 2
    { gps_simpleStart, (4*1024), "GPS SIMPLE", 3 }, // ID = 3
    { user_task, (4*1024), "USER_TASK", 2 }, // ID = 4
    /* Aiding mode Task */
    { Task_GPS_AIDING_AEE, GPS_CORE_AIDING_AEE_STACK_SIZE, "AEE", 1 }, // ID = 5
    { 0, 0, 0, 0 }
};

void adl_task( void )
{
    TRACE(( 1, "adl_task is started, task : %d", adl_ctxGetID() ));
}

void user_task( void )
{
    TRACE(( 1, "adl_task is started, task : %d", adl_ctxGetID() ));
}

static void gps_simplePvtHandler(gps_pvtInfo_t * pPvtInfo)
{
    ascii PvtStr[250];
    // GPSPVT: 0
    u16 latDeg, latMin;
    double latSec;
    u16 longDeg, longMin;
    double longSec;
    double dummyValue;
    /* Convert Latitude to llll.ll format -> degrees|minutes.decimal */
    latDeg = (u16)pPvtInfo->tPositionInfo.latitude;
    dummyValue = fabs( (pPvtInfo->tPositionInfo.latitude - latDeg) * 60.0 );
    latMin = (u16)dummyValue;
    latSec = fabs( (dummyValue - latMin) * 60.0 );
    /* Convert Longitude to yyyyy.yy format -> degrees|minutes.decimal */
    longDeg = (u16)pPvtInfo->tPositionInfo.longitude;
    dummyValue = fabs( (pPvtInfo->tPositionInfo.longitude - longDeg) * 60.0 );
    longMin = (u16)dummyValue;
    longSec = fabs( (dummyValue - longMin) * 60.0 );
    sprintf( PvtStr, sizeof(PvtStr)
// +GPSPVT: 0,08:17:32 ,27/04/2010 ,3D FIX, N 48°34'52.90'' ,E 02°21'58.65'' ,+0010m
    , "\r\n+GPSPVT: 0,%02d:%02d:%02d,%02d/%02d/%04d,%s,%c %02d%c%02d' %05.21f\", %c %02d%c%02d' %05.21f\", %c%04.0fm\r\n"
    , pPvtInfo->tTimeDateInfo.hours
    , pPvtInfo->tTimeDateInfo.minutes
    , pPvtInfo->tTimeDateInfo.seconds
    , pPvtInfo->tTimeDateInfo.day
    , pPvtInfo->tTimeDateInfo.month
    , pPvtInfo->tTimeDateInfo.year
    , pPvtInfo->tPositionInfo.eFixState == GPS_FIX_NO_POS ? "NO FIX":pPvtInfo->tPositionInfo.eFixState == GPS_FIX_2D ?
    "2D FIX" : pPvtInfo->tPositionInfo.eFixState == GPS_FIX_DIFF_2D ? "2D FIX": pPvtInfo->tPositionInfo.eFixState == GPS_
    FIX_3D ? "3D FIX": pPvtInfo->tPositionInfo.eFixState == GPS_FIX_DIFF_3D ? "3D FIX" : pPvtInfo->tPositionInfo.eFixStaE
    == GPS_FIX_ESTIMATED ? "ES FIX" : "UN FIX"
    , pPvtInfo->tPositionInfo.latitude >= 0.0 ? 'N' : 'S'
    , latDeg
    , , // , °\
    , latMin
    , latSec
    , pPvtInfo->tPositionInfo.longitude < 0.0 ? 'W' : 'E'
    , longDeg
    , \ \ // \ , °\
    , longMin
    , longSec
    , pPvtInfo->tPositionInfo.altitudeMsl >= 0 ? '+' : '-'
    , fabs(pPvtInfo->tPositionInfo.altitudeMsl));
    // Send PVT sentences
    adl_atSendResponsePort ( ADL_AT_UNS , ADL_PORT_NONE, (ascii*)PvtStr);
}

static void gps_simpleBoostModeHandler(bool bBoostMode)

```

Listing 1. c.d.

```

{
    s32 l_s32_retStatus = OK;
    TRACE(( 2, "[GPS_SIMPLE] gps_simpleBoostModeHandler %d", bBoostMode));
    // Update the CPU clock mode
    if (bBoostMode == TRUE)
    {
        l_s32_retStatus = adl_vsSetClockMode( s32VsHandle, ADL_VS_MODE_BOOST );
    }
    else
    {
        l_s32_retStatus = adl_vsSetClockMode( s32VsHandle, ADL_VS_MODE_STANDARD );
    }

    if (l_s32_retStatus < 0)
    {
        TRACE(( 2, "[gps_simpleBoostModeHandler] Error from adl_vsSetClockMode [%d]", l_s32_retStatus));
    }
    return;
}

static void gps_simpleInfoHandler( gps_infoEvent_t * pPosEvent )
{
    u32 ttff = 0;
    switch ( pPosEvent->eEventType )
    {
        case GPS_POS_FIX_LOST_EVENT:
            TRACE(( 1, "[GPS_SIMPLE] gps_simpleInfoHandler: GPS_POS_FIX_LOST_EVENT"));
            break;
        case GPS_POS_FIX_2D_EVENT:
            TRACE(( 1, "[GPS_SIMPLE] gps_simpleInfoHandler: GPS_POS_FIX_2D_EVENT"));
            break;
        case GPS_POS_FIX_3D_EVENT:
            gps_infoGetTTFF (&ttff);
            TRACE(( 1, "[GPS_SIMPLE] gps_simpleInfoHandler: GPS_POS_FIX_3D_EVENT"));
            TRACE(( 1, "[GPS_SIMPLE] TTFF = %d", (ttff/1000)));
            break;
        case GPS_POS_FIX_ESTIMATED_EVENT:
            TRACE(( 1, "[GPS_SIMPLE] gps_simpleInfoHandler: GPS_POS_FIX_ESTIMATED_EVENT"));
            break;
        case GPS_POS_FIX_INVALID_EVENT:
            TRACE(( 1, "[GPS_SIMPLE] gps_simpleInfoHandler: GPS_POS_FIX_INVALID_EVENT"));
            break;
        default:
            TRACE(( 1, "[GPS_SIMPLE] gps_simpleInfoHandler: Internal Error!"));
            break;
    }
}

static void gps_simpleEventHandler( gps_event_e eEvent, void* pEventData )
{
    gps_status_e status;
    if ( eEvent == GPS_INIT_EVENT )
    {
        TRACE(( 1, "[GPS_SIMPLE] event: GPS_INIT_EVENT"));
        gps_pvtSetOpts( GPS_OPT_PVT_HANDLER, gps_simplePvtHandler
            , GPS_OPT_PVT_RATE, 1000, GPS_OPT_END);
        gps_infoSetOpts( GPS_OPT_INFO_HANDLER, gps_simpleInfoHandler
            , GPS_OPT_END);
        gps_coreSetOpts( GPS_OPT_CORE_LNA_EXT
            , GPS_LNA_EXTERNAL, GPS_OPT_END );
        gps_coreSetOpts( GPS_OPT_CORE_BOOST_MODE_HANDLER
            , gps_simpleBoostModeHandler, GPS_OPT_END );
        gps_coreSetOpts( GPS_OPT_CORE_AIDING_MODE, GPS_AIDING_AEE, GPS_OPT_END);
        status = gps_start(GPS_HOT_START);
        TRACE((1, "Start Status = %d", status));
    }
    Else TRACE((1, "[GPS_SIMPLE] event: %d", eEvent));
}

void gps_simpleStart( void )
{
    TRACE(( 1, "[gps_simpleStart]"));
    s32VsHandle = adl_vsSubscribe();
    gps_init(GPS_HWTYPE_XM0110
        , gps_simpleEventHandler
        , &ioConfigXM0110);
}

```

zmniejszyło rozmiar kodu, ale przy zachowaniu jego pełnej funkcjonalności.

Pierwsze, na co należy zwrócić uwagę analizując program z list. 1, to fakt, że aplikacja składa się z kilku wątków. Dwa z nich *Task_GPS_COM* oraz *Task_GPS_CORE* to wątki należące do biblioteki *Location Plugin Package* i odpowiedzialne są za poprawną obsługę modułu XM0110. Mają one, poza wątkiem *adl_task*, najwyższy priorytet w systemie i ich wywołanie jest konieczne do obsługi modułu XM0110. Wątek *adl_task*, o ile nie wystąpi taka potrzeba powinien pozostać pusty. Kod użytkownika najlepiej

umieścić w wątku o priorytecie niższym niż wątki do obsługi GPS, czyli w naszym przykładzie jest to *user_task*.

Wątkiem o najniższym priorytecie jest *Task_GPS_AIDING_AEE*, który odpowiedzialny jest za liczenie danych wspomagających, które przyspieszą synchronizację GPS podczas następujących startów. Opisem funkcji wspomagającej zajmijemy się w dalszej części artykułu.

Właściwa inicjalizacja GPS następuje natomiast w wątku *gps_simpleStart*. Tu następuje subskrypcja do serwisu *varispeed* (wykorzystywanego później) oraz wywołanie

funkcji *gps_init()*. Argumentami wywołania tej funkcji są: typ modułu (obecnie obsługiwany jedynie XM0110), funkcja zdarzeń, a także struktura inicjalizująca. Struktura ta opisuje konfigurację sprzętową połączenia pomiędzy modułami (możliwe sposoby połączenia przedstawione zostały w poprzednim odcinku). Zawiera ona informacje o wybranym interfejsie komunikacyjnym, wykorzystywanym do połączenia modułu GSM z modułem XM0110; numery linii GPIO wykorzystane jako linia ON/OFF i reset dla modułu XM0110, a także skąd pochodzi źródło sygnału 32 kHz dla modułu GPS.

Tabela 1. Porównanie czasów synchronizacji GPS

TTF (50%, -130 dBm)	Z liczeniem AEE	Bez liczenia AEE
Cold Start (commanded)	<20 s	<35 s
Warm Start (commanded)	<10 s	<35 s

Tabela 2. Pobór energii w trybie Low Power Navigation Mode

Tryb pracy	Wielkość okna GPS RF	Prąd średni XM0110	Dokładność ustalania pozycji przez GPS (-130 dBm) (50%)
Full Power navigation	Continuous	32 mA	<1,5 m
1 Hz Medium Power navigation	600 ms	16 mA	<2 m
1 Hz Low Power navigation	200 ms	11 mA	<5 m
1 Hz Very Low Power navigation	100 ms	7 mA	<7 m

Tabela 3. Zestawienie trybów oszczędzania energii dla modułu XM0110

Tryb pracy		Prąd średni (LNA wewnętrzny)	Prąd średni (LNA zewnętrzny)	Jednostka
Push to Fix	GPS Off	<5		μA
	GPS Hibernate	26		μA
	GPS Idle	1,1	1,0	mA
1 Hz Navigation	Very Low Power Navigation	8	7	mA
	Low Power Navigation	12	11	mA
	Medium Power Navigation	18	16	mA
	Full Power Navigation	36	32	mA
Acquisition		45	41	mA

Po wywołaniu funkcji `gps_init()` oczekujemy na pojawienie się zdarzenia `GPS_INIT_EVENT`. Po jego wystąpieniu dokonujemy ustawienia kilku dodatkowych parametrów. Na początku wskazujemy funkcję, która będzie przechwytywała informacje PVT (Position, Velocity, Time) z określoną w milisekundach częstotliwością. Wskazany handler (w naszym przypadku `gps_simplePvtHandler`) będzie otrzymywał co określony czas strukturę (`gps_pvtInfo_t`) zawierającą informacje o pozycji, prędkości, czasie, itd. W funkcji handlera dane ze struktury są formowane w finalną ramkę PVT.

Istnieje również możliwość generowania ramek NMEA. Może mieć to zastosowanie, gdy informacje o pozycji potrzebujemy przesłać do innego urządzenia przyjmującego właśnie ramki NMEA lub wizualizować informacje o pozycji na PC za pomocą aplikacji czytającej ramki NMEA. Aby generować ramki NMEA należałoby skorzystać z funkcji:

```
gps_nmeaGetOpts(gps_nmeaHandler,
GPS_NMEA_GGA_EN | GPS_NMEA_RMC_EN,1000);
```

Kolejnym parametrem, który podajemy to funkcja zdarzeń dla informacji o stanie synchronizacji GPS. Po wystąpieniu zdarzenia `GPS_POS_FIX_3D_EVENT` licznicy jest czas TTFF (Time To First Fix) jaki minął do osiągnięcia pełnej synchronizacji. Tu również można użyć funkcji `gps_infoGetDetailedPos()`, która zwraca informacje o pozycji w strukturze `gps_pvtInfo_t`. Korzystając z tej funkcji otrzymujemy informacje o pozycji na żądanie.

Można wtedy zrezygnować z funkcji zdarzeniowej – `gps_simplePvtHandler()` – przesyłając dane o pozycji w regularnych odstępach.

Ustawienie zewnętrznego lub wewnętrznego wzmacniacza LNA konieczne jest do skonfigurowania typu anteny GPS z jakiej będziemy korzystać. W naszym przypadku ustawiamy zewnętrzny LNA, czyli antenę aktywną.

W funkcji `gps_simpleStart()` subskrybowaliśmy się do serwisu VariSpeed. Teraz za pomocą funkcji `gps_coreSetOpts(GPS_OPT_CORE_BOOST_MODE_HANDLER, gps_simpleBoostModeHandler, GPS_OPT_END)`; wskazujemy funkcję, która w zależności od parametru typu bool zmienia częstotliwość taktowania procesora modemu GSM na 26 MHz lub 104 MHz.

Przedostatnim parametrem, który ustawiamy jest włączenie funkcji wspomagającej synchronizację GPS. Nie jest to obowiązkowe, ale w większości przypadków znacznie przyspiesza czas synchronizacji GPS. Generalnie sposobów wspomagania GPS, czyli metod osiągnięcia synchronizacji szybszej od standardowej jest wiele. W obecnej wersji `Location Plugin` jest dostępna tylko metoda polegająca na liczeniu Efemeryd Rozszerzonych (`Extended Ephemeris`). W kolejnych wersjach plugin'u jest planowana implementacja kolejnych metod wspomagania, jak polegające na podaniu poprawki czasu oraz możliwość pobrania danych dla Efemeryd Rozszerzonych ze wskazanego serwera. Wspomniany wcześniej wątek o najniższym

priorytecie, czyli `Task_GPS_AIDING_AEE` w chwilach bezczynności procesora dokonuje obliczeń Efemeryd Rozszerzonych dla poszczególnych satelitów będących aktualnie w polu widzenia. Dane te w odróżnieniu od zwykłych efemeryd są ważne 3 dni, ale ich wyliczenie wymaga zaangażowania mocy obliczeniowej procesora. W trakcie tych obliczeń procesor jest przełączany na częstotliwość taktowania wynoszącą 104 MHz. Wyliczone dane są przechowywane w pamięci nieulotnej i pomagają uzyskać szybszą synchronizację GPS podczas następnego startu. Dzieje się tak dlatego, że standardowo na zdekodowanie danych dla Efemeryd z sygnału GPS potrzeba 24 sekundy czasu, a w przypadku użycia AEE dane te są dostępne od razu. Przykładowe różnice czasu synchronizacji podano w **tabeli 1**.

Ostatnią funkcją, którą wywołujemy pod zdarzeniem `GPS_INIT_EVENT`, jest `gps_start()`. Powoduje ona uruchomienie wszystkich bloków układu XM0110 i właściwe rozpoczęcie procesu uzyskiwania pozycji. Po wywołaniu tej funkcji powinniśmy odebrać zgłoszenie zdarzenia nr 2 (`GPS_START_EVENT`) w funkcji `gps_simpleEventHandler()` świadczące o poprawnym starcie modułu. Funkcja `gps_start()` może być wywoływana z następującymi parametrami:

- `GPS_HOT_START` – uruchomienie ze wszystkimi danymi nawigacyjnymi zapisanymi w pamięci. Poprzednie uruchomienie powinno się zakończyć funkcją `gps_sleep()`.
- `GPS_WARM_START` – uruchomienie ze wszystkimi danymi nawigacyjnymi poza danymi efemeryd. Poprzednie uruchomienie powinno się zakończyć funkcją `gps_sleep()`.
- `GPS_COLD_START` – uruchomienie bez zapamiętanych danych nawigacyjnych. Wykorzystana jest jedynie zapamiętana poprawka czasu.
- `GPS_FACTCOLD_START` - start fabryczny (z danymi zapisanymi na etapie produkcji).

Po uruchomieniu aplikacji po 3...5 s powinny pojawić się ramki PVT. Początkowo będą one zawierały same zera. Następnie, w miarę dekodowania sygnału GPS, będą wypełniane pola z danymi czasu oraz daty, aż wreszcie otrzymamy również dane z pozycją GPS (**rysunek 3**).

Podsumowując opisaną powyżej aplikację trzeba również wspomnieć o zasobach zużywanych przez bibliotekę `Location Plugin Package`. Jeśli chodzi o moc obliczeniową modułu to praca biblioteki pochłania:

- ok. 25% mocy obliczeniowej CPU jeśli wykorzystujemy do połączenia interfejs UART,
- ok. 30% mocy obliczeniowej CPU jeśli wykorzystujemy do połączenia interfejs I²C,

- dodatkowo podczas obliczania Efemeryd Rozszerzonych przez proces AEE wykorzystywane jest do 85% mocy obliczeniowej CPU.

Natomiast zużycie pamięci wygląda następująco:

pamięć ROM: 679 kB (rozmiar biblioteki w skompilowanym pliku wynikowym),
 pamięć RAM: 419 kB (z procesem AEE),
 pamięć RAM: 326 kB (bez procesu AEE),
 pamięć Flash: 106 kB (z procesem AEE),
 pamięć Flash: 20 kB (bez procesu AEE).

Tryby oszczędzania energii

Low Power Navigation Mode

Z punktu wykorzystania modułu XM0110 w rozwiązaniach mobilnych dość istotną kwestią staje się zużycie energii oraz tryby jej oszczędzania. Standardowo moduł pracuje w trybie nawigacji, gdzie dane o pozycji są aktualizowane z częstotliwością 1 Hz. Podczas szukania satelitów (*acquisition*) jest pobierany prąd o natężeniu ok. 41 mA, natomiast po uzyskaniu synchronizacji ok. 32 mA. W celu zmniejszenia poboru prądu przez moduł GPS można go przełączyć w tryb *Low Power Navigation Mode*. W tym trybie moduł również aktualizuje dane o pozycji z częstotliwością 1 Hz, ale zmniejszane jest okno czasowe, czyli czas, w którym moduł odbiera sygnał z satelitów. Dla tego trybu są dostępne następujące ustawienia: *Medium Power*, *Low Power* oraz *Very Low Power*. Każda z tych opcji charakteryzuje się różną długością okna czasowego.

Sposób włączenie trybu *Low Power Navigation Mode* w Open AT wygląda następująco:

```
gps_init(...)
```

```
gps_coreSetOpts(GPS_OPT_CORE_RUNNING_POWER_MODE, X, GPS_OPT_END);
```

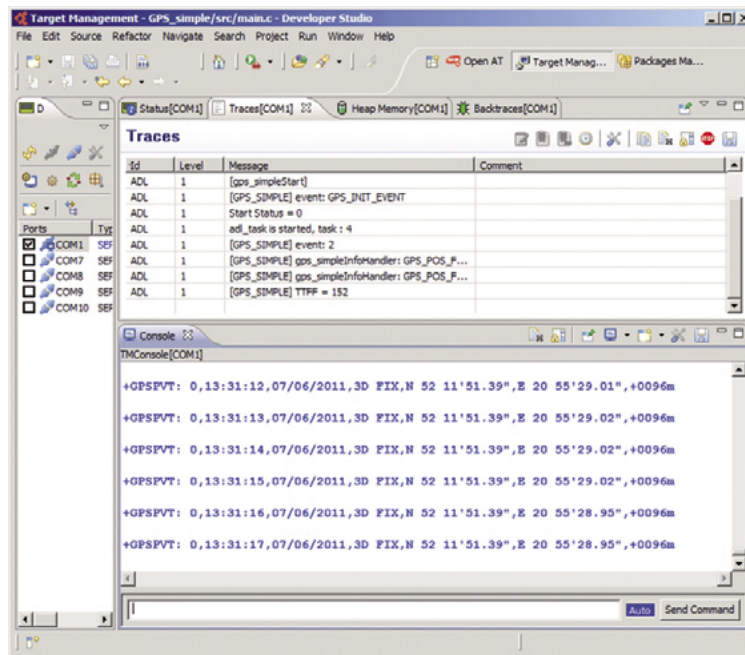
```
gps_start(...)
```

X może przyjąć następujące wartości:

- 0: Full power navigation mode
- 1: Medium power navigation mode
- 2: Low power navigation mode
- 3: Very low power navigation mode

Pobory prądów dla trybu *Low Power Navigation Mode* w zależności od wartości parametru X przedstawiono w tabeli 2.

Można zauważyć, że skrócenie okna czasowego zmniejsza zużycie energii przez moduł. Dzieje się to kosztem zmniejszenia



Rysunek 3. Efekt działania aplikacji z listingu 1

dokładności ustalonej pozycji, co również pokazano w tab. 2. Wprowadzenie modułu w tryb *Low Power Navigation Mode* nie nastąpi dopóty, dopóki moduł nie uzyska 3D fix lub sygnał z 4 najlepiej widocznych satelitów będzie zbyt słaby.

W przyszłej wersji plugin'u zostanie zaimplementowany również tryb pozwalający na obniżenie częstotliwości wyliczania pozycji z 1 Hz do niższych częstotliwości w zakresie 0,5 Hz do 0,1 Hz. Pozwoli to na dalsze obniżenie energii pobieranej przez moduł.

Idle Mode, Hibernate Mode

Jeśli w danej chwili informacje o pozycji nie są nam potrzebne, to możemy przełączyć moduł GPS w tryb IDLE. W tym trybie informacje o pozycji nie są dostępne, a poziom zużywanej energii znacznie zmniejszony. Moduł zachowuje częściową funkcjonalność automatycznie monitorując widoczne satelity w pewnych odstępach czasowych, co daje możliwość osiągnięcia możliwie niskiego czasu TTFF po ponownym wywołaniu funkcji *gps_start()*.

Wejście do trybu *Idle Mode* uzyskujemy dzięki uruchomieniu funkcji *gps_sleep(GPS_IDLE_MODE)*;

Tryb *Hibernate Mode* zapewnia jeszcze większą oszczędność energii. W tym trybie zachowane zostają jedynie informacje o poprawce czasu, co nie gwarantuje już tak szybkiego czasu ponownej synchronizacji. Wejście do trybu *Idle Mode* następuje po wywołaniu funkcji *gps_sleep(GPS_HIBERNATE_MODE)*;

Rozsądne stosowanie wymienionych powyżej trybów pozwala na znaczne oszczędności zużywanej przez moduł energii. Zestawienie opisanych trybów wraz z wartościami prądów umieszczono w tabeli 3.

Aby przetestować działanie przedstawionych trybów można w łatwy sposób zmodyfikować program z list. 1 o kilka komend AT. Zmodyfikowany program zostanie dołączony wraz z dokumentacją software modułu XM0110 na płycie dołączonej do numeru.

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 opisanych produktów oraz zapewnia pełne wsparcie techniczne.

Adrian Chranowski
 Acte Sp. z o.o.

REKLAMA

Minimoduł z ATMEGA8

AVT1622

Więcej informacji:



www.sklep.avt.pl

