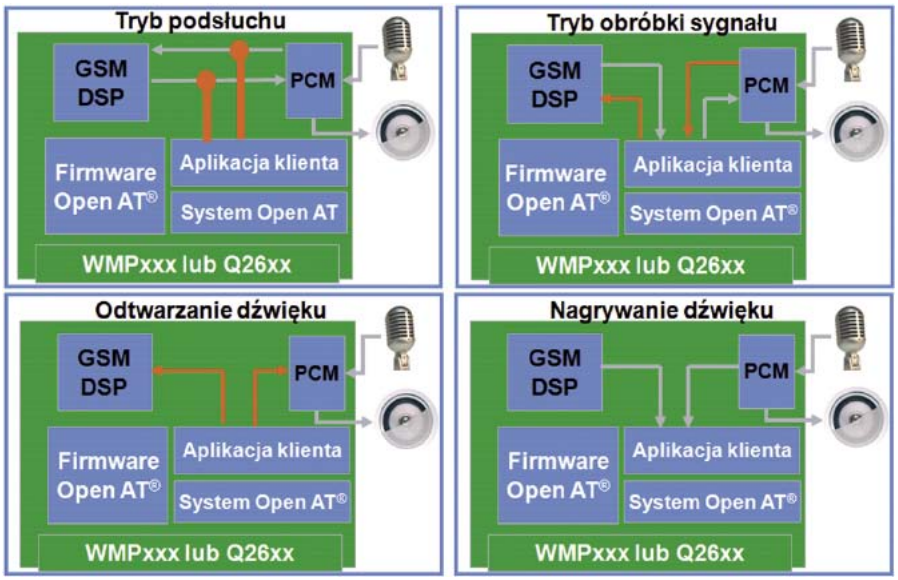
Technologia GSM w elektronice (11)

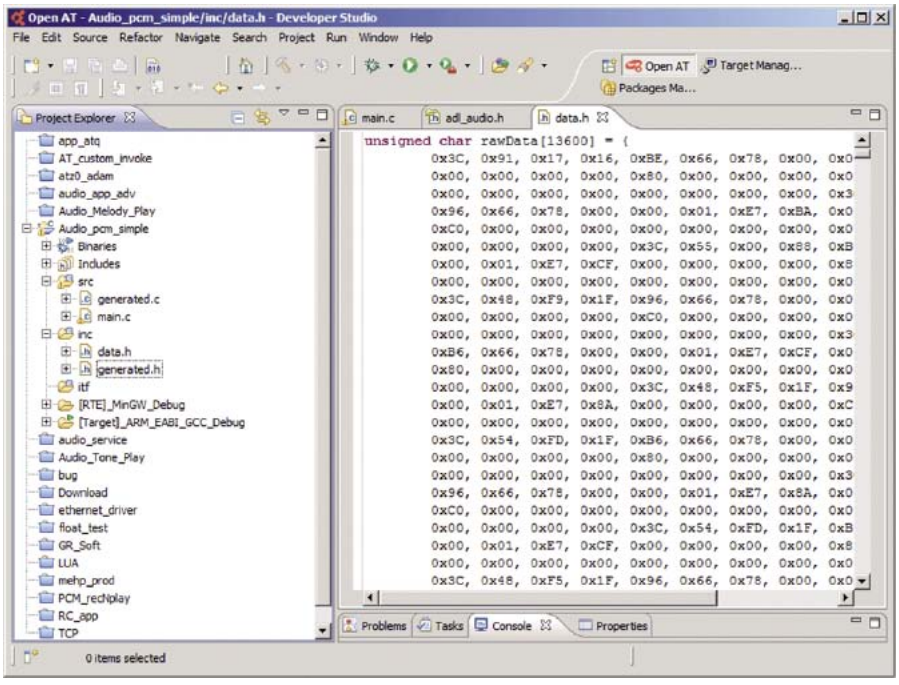
Open AT – Serwis Audio

W tym odcinku kursu dotyczącym programowania modułów GSM Sierra Wireless przedstawimy możliwości systemu Open AT w zakresie obsługi funkcji audio. Częściowo wykorzystamy wiedzę na temat przerwań systemowych prezentowaną w poprzednim odcinku kursu.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 12147, pass: 2e7u6a2a
• poprzednie części kursu



Rysunek 1. Strumień audio – możliwe zastosowania



Rysunek 2. Widok tablicy z próbkami dźwięku

System operacyjny Open AT pracujący na programowalnych urządzeniach Sierra Wireless daje programiście liczne możliwości wykorzystania funkcji audio w swojej aplikacji. Mowa tu nie tylko o generowaniu prostych dźwięków i melodii, ale również o tak zaawansowanych funkcjach, jak generowanie i detekcja tonów DTMF, a także odtwarzanie i rejestrowanie dźwięku w formacie PCM lub AMR. Zacznijmy jednak od rzeczy najprostszych, czyli od odtwarzania pojedynczych tonów.

Zgodnie z obowiązującą w Open AT zasadą zdarzeniowości należy najpierw dokonać subskrypcji do serwisu audio. Do tego celu wykorzystujemy funkcję *adl_audioSubscribe()*. Jako parametry funkcji podajemy: używany zasób sprzętowy, funkcję zdarzeniową callback oraz parametr mówiący, czy zasób sprzętowy może być wywłaszczany (na przykład buzzer może być wywłaszczony przez system, aby odegrać melodyjkę dzwonka w momencie przychodzącego połączenia GSM). W systemie Open AT mamy do dyspozycji następujące zasoby sprzętowe:

- *ADL_AUDIO_SPEAKER* – aktualnie wybrany głośnik jako urządzenie wyjściowe. Parametry głośnika (np. domyślny głośnik, wzmocnienie, echo) możemy modyfikować za pomocą systemowych komend AT.
- *ADL_AUDIO_BUZZER* – buzzer jako urządzenie wyjściowe. Za jego pomocą można odegrać jedynie proste melodie lub pojedyncze tony.
- *ADL_AUDIO_MICROPHONE* – mikrofon jako urządzenie wejściowe. Parametry jak w przypadku głośnika można modyfikować komendami AT.
- *ADL_AUDIO_VOICE_CALL_RX* – tor audio (urządzenie wejścia) dla trwającego połączenia głosowego GSM.
- *ADL_AUDIO_VOICE_CALL_TX* – tor audio (urządzenie wejścia) dla trwającego połączenia głosowego GSM.

Przykładową aplikację generującą pojedynczy ton słyszalny w głośniku przedstawiono na **listingu 1**.

Zadaniem aplikacji jest odegranie tonu o częstotliwości 300 Hz, ze wzmocnieniem –10 dB oraz o czasie trwania 500 ms. W przypadku głośnika jest możliwe odegranie równocześnie dwóch tonów, np.: *Ret = adl_audioTonePlay(handle, 300, –10, 1000, –5, 500);*

Nie jest to możliwe w przypadku buzzera, ponieważ akceptuje on tylko jedną częstotliwość.

Oprócz generowania pojedynczych tonów, jest również możliwość odgrywania konkretnych melodii na podstawie zapisu nutowego. Przykład takiej aplikacji przedstawiono na **listingu 2**.

W powyższym przykładzie melodia została zawarta w postaci zapisu nutowego przez podanie wartości nuty, oktawy oraz czasu jej trwania. Argumentami funkcji *adl_audioMelodyPlay()* oprócz uchwytu i tablicy z zapisem nutowym są także, tempo (bpm), liczba cykli odtwarzania melodii oraz poziom głośności (dB).

Zarówno w przypadku pojedynczego tonu, jak i całej melodii jako urządzenie wyjściowe możemy przyjąć jedynie speaker albo buzzer. Dużo większe możliwości dają serwisy *audio stream* pozwalające na odtwarzanie lub nagrywanie dźwięków w formacie PCM lub ARM. Tu jako urządzenie wejścia/wyjścia możemy wykorzystać zarówno interfejsy lokalne w postaci mikrofonu (we) lub głośnika (wy), jak również strumień dźwięku aktualnie trwającego połączenia GSM. Takie możliwości pozwalają na nagrywanie bądź odtwarzanie dźwięku urządzenia, które uzyskało połączenie głosowe z naszym modułem GSM.

Przykładową aplikację wykorzystującą możliwości odtwarzania strumienia audio zamieszczono na **listingu 3**.

Aplikacja tworzy dodatkową komendę *at+graj*. Wywołana z parametrem 0 spowoduje odtworzenie strumienia audio przez głośnik lub słuchawki. Jeśli komenda ta zostanie wywołana z innym parametrem, np. *at+graj=1*, to aplikacja najpierw nawiąże połączenie audio ze wskazanym numerem telefonu, a po odebraniu połączenia przez drugą stronę zacznie odtwarzać dźwięk.

Na potrzeby aplikacji wykorzystano zarówno przerwanie niskiego poziomu, jak i przerwanie wysokiego poziomu, więc w formatce generated.c należy zadeklarować wartość stosu dla procesów obsługi przerwań.

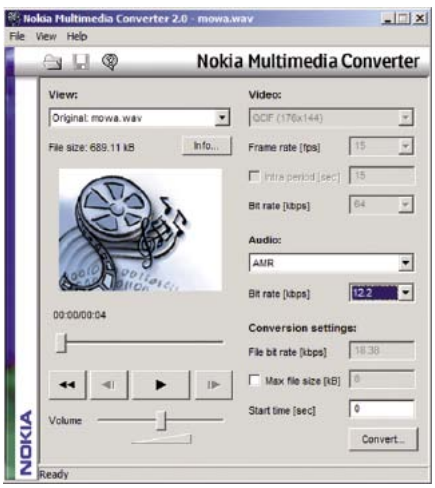
W obsłudze przerwania LowLevel cyklicznie do bufora *StreamBuffer* ładowane są kolejne próbki dźwiękowe. Jeśli już zo-

Listing 1. Przykładowy program generujący pojedynczy ton

```
#include "adl_global.h"
#include „generated.h"
s32 handle; // audio resource handle

void MyAudioEventHandler(s32 audioHandle, adl_audioEvents_e Event)
{
  s32 Ret;
  switch ( Event)
  {
    case ADL_AUDIO_EVENT_NORMAL_STOP :
      TRACE (( 1, " Audio handle %d : stop ", audioHandle ));
      Ret = adl_audioUnsubscribe ( handle );//unsubscribe to audioService
      break;
    case ADL_AUDIO_EVENT_RESOURCE_RELEASED :
      break;
    default : break;
  }
}

void main_task ( void ){
  s32 Ret;
  handle = adl_audioSubscribe ( ADL_AUDIO_SPEAKER, MyAudioEventHandler,
  ADL_AUDIO_RESOURCE_OPTION_FORBID_PREEMPTION );
  Ret = adl_audioTonePlay( handle, 300, -10, 0, 0, 500 ); // 300 Hz, -10 dB,
500 ms
}
```

Listing 2. Przykład aplikacji do odtwarzania melodii na podstawie zapisu nutowego

```
#include "adl_global.h"
#include „generated.h"
s32 handle;
#define oktawa 5
static const u16 Melody_Sample[ ]=
{  ADL_AUDIO_NOTE_DEF( ADL_AUDIO_E, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_FS, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_G, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_A, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_B, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_G, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_B, oktawa, ADL_AUDIO_QUARTER ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_AS, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_FS, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_AS, oktawa, ADL_AUDIO_QUARTER ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_A, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_F, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_A, oktawa, ADL_AUDIO_QUARTER ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_E, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_FS, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_G, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_A, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_B, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_G, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_B, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_E, (oktawa+1), ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_D, (oktawa+1), ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_B, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_G, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_B, oktawa, ADL_AUDIO_HEIGHTH ),
   ADL_AUDIO_NOTE_DEF( ADL_AUDIO_D, (oktawa+1), ADL_AUDIO_HALF ),
   NULL
};

void MyAudioEventHandler ( s32 audioHandle, adl_audioEvents_e Event )
{
  s32 Ret;
  switch ( Event){
  case ADL_AUDIO_EVENT_NORMAL_STOP :
    TRACE (( 1, „ Audio handle %d : stop „, audioHandle ));
    Ret = adl_audioUnsubscribe ( handle );
    break;
  case ADL_AUDIO_EVENT_RESOURCE_RELEASED :
    break;
  default : break;
  }
}
void main_task ( void )
{
  s32 Ret;
  handle = adl_audioSubscribe ( ADL_AUDIO_SPEAKER, MyAudioEventHandler ,
  ADL_AUDIO_RESOURCE_OPTION_FORBID_PREEMPTION );
  Ret = adl_audioMelodyPlay( handle, Melody_Sample, 100, 2, 0); //
tempo:100,0dB,odt:2x
}
```



Rysunek 3. Konwersja do formatu AMR

stały pobrane wszystkie możliwe próbki, to funkcja obsługi zwraca wartość true. To z kolei spowoduje wykonanie przerwania *HighLevel*, gdzie następuje zatrzymanie odtwarzania oraz zwolnienie zasobów pamięci.

Wywołanie odtwarzania przez lokalny głośnik/słuchawki różni się od wywołania do toru GSM w zasadzie tylko jednym pa-

Listing 3. Przykład aplikacji do odtwarzania strumienia audio

```
#include "adl_global.h"
#include "generated.h"
#include "data.h"

#define phone_nb "+48xxxxxxxxx"  // wpisz wlasciwy numer telefonu
// audio resource handle
s32 handle;
u32 dataptr = 0;
s32 BufferSize=32;
void * StreamBuffer; // audio stream buffer

s32 audio_IrqHandle_L,audio_IrqHandle_H;

// Low level interrupt handler
bool MyHghLevelIRQHandler ( adl_irqID_e Source, adl_irqNotificationLevel_e Notification_Level, adl_irqEventData_t * Data
)
{
  s32 RetVal;
  TRACE((1,"inside low IRQ handler: STOP"));
  RetVal = adl_audioStop(handle);
  TRACE((1,"HghLevel : RetVal = %d", RetVal));
  adl_audioUnsubscribe(handle);
  adl_memRelease(StreamBuffer);
  StreamBuffer= NULL;
}
bool MyLowLevelIRQHandler ( adl_irqID_e Source, adl_irqNotificationLevel_e Notification_Level, adl_irqEventData_t * Data
)
{
  // copy PCM sample to play
  TRACE((1,"inside low IRQ handler, dataptr = %d ",dataptr));
  if ((dataptr + BufferSize) < sizeof (rawData)) {
    wm_memcpy(StreamBuffer, &rawData[dataptr], BufferSize);
    dataptr += BufferSize;
    ( ( adl_audioStream_t * )Data->SourceData )->BufferReady = TRUE; // Set BufferReady flag to TRUE
    return FALSE;
  }
  else {
    ((adl_audioStream_t * )Data->SourceData )->BufferReady = FALSE;
    return TRUE;  // koniec : wykonaj MyHghLevelIRQHandler
  }
}

void MyAudioEventHandler (s32 audioHandle, adl_audioEvents_e Event)
{
  TRACE (( 1, "Inside the AudioEventHandler: Event :: %d", Event ));
  return;
}

void play_stream(){
  s32 Ret;
  Ret = adl_audioSetOption(handle, ADL_AUDIO_AMR_MIXED_VOICE, TRUE);
  Ret = adl_audioSetOption(handle, ADL_AUDIO_AMR_SPEECH_CODEC_RATE, ADL_AUDIO_AMR_RATE_12_20);
  // Memory allocation
  Ret = adl_audioSetOption ( handle,ADL_AUDIO_AMR_BUFFER_SIZE, BufferSize );
  TRACE((1,"Bufer size = %d",BufferSize));
  StreamBuffer = adl_memGet( BufferSize ); // release memory after audio stream playing
  Ret = adl_audioStreamPlay( handle, ADL_AUDIO_AMR, audio_IrqHandle_L, audio_IrqHandle_H, StreamBuffer);
}

s8 Call_Handler ( u16 Event, u32 Call_ID ){
  TRACE((1,"Call_Handler:Event = %d",Event));
  if(Event== ADL_CALL_EVENT_SETUP_OK){  //polaczenie nawiazane
    handle = adl_audioSubscribe ( ADL_AUDIO_VOICE_CALL_TX, MyAudioEventHandler ,ADL_AUDIO_RESOURCE_OPTION_FORBID_
PREEMPTION );
    play_stream();
  }
}

void Fun_play(adl_atCmdPreParser_t * param) {

  s8 voice_dest;
  ascii * parametr;
  if (param->Type == ADL_CMD_TYPE_PARA){
    parametr = ADL_GET_PARAM ( param, 0 );
    voice_dest = wm_atoi(parametr);
    if(voice_dest == 0){  // jeśli 0 graj na speaker
      handle = adl_audioSubscribe ( ADL_AUDIO_SPEAKER, MyAudioEventHandler ,ADL_AUDIO_RESOURCE_OPTION_FORBID_PREEMPTION
);
      play_stream();
    }
    else{  //jeśli inne niz 0 zadzwon
      adl_callSetup(phone_nb, ADL_CALL_MODE_VOICE);
    }
  }
}

void main_task ( void )
{
  TRACE((1,"Sample size = %d",sizeof(rawData)));
  audio_IrqHandle_L = adl_irqSubscribe ( MyLowLevelIRQHandler, ADL_IRQ_NOTIFY_LOW_LEVEL, ADL_IRQ_PRIORITY_LOW_LEVEL,
ADL_IRQ_OPTION_AUTO_READ );
  audio_IrqHandle_H = adl_irqSubscribe ( MyHghLevelIRQHandler, ADL_IRQ_NOTIFY_HIGH_LEVEL, ADL_IRQ_PRIORITY_HIGH_LEVEL,
ADL_IRQ_OPTION_AUTO_READ );
  adl_atCmdSubscribe(„at+start", (adl_atCmdHandler_t *) Fun_play, ADL_CMD_TYPE_PARA | 0x11);
  adl_callSubscribe ( Call_Handler );
}
```

rametrem funkcji *adl_audioSubscribe()*. Dla toru lokalnego jest to *ADL_AUDIO_SPEAKER*, natomiast dla toru GSM – *ADL_AUDIO_VOICE_CALL_TX*. Pozostały kod odpowiedzialny za odtworzenie dźwięku został umieszczony w funkcji *play_stream()*. Tu ustawiamy kilka parametrów, takich jak:

– *ADL_AUDIO_AMR_MIXED_VOICE* – parametr określający, czy dźwięk z mikrofonu ma być miksowany z odtwarzanym strumieniem (wartość true), czy ma być wyciszony podczas odtwarzania (false).
– *ADL_AUDIO_AMR_SPEECH_CODEC_RATE* – parametr określający jakość próbkowania dźwięku w formacie AMR. Do wyboru mamy kilka wartości bitrate dla AMR i kilka wartości dla AMR-WB (Wide Band). Dźwięk w formacie AMR-WB jest lepszej jakości ze względu na 16 kHz próbkowanie, ale może być odtwarzany jedynie w kierunku układu lokalnego. Wykorzystując odtwarzanie w tym formacie, należy również przestawić układ lokalny na odtwarzanie przy próbkowaniu 16 kHz za pomocą komendy *AT+SPEAKER=12* dla interfejsu wyjściowego *SPEAKER1* lub *AT+SPEAKER=13* dla *SPEAKER2*.

Kolejnym krokiem jest zaalokowanie pamięci dla bufora *StreamBuffer*. Rozmiar bufora dla dźwięku w formacie AMR powinien być równy rozmiarowi zajmowanemu przez całą próbkę dźwiękową, którą zamierzamy odtworzyć lub 20 ms ramce dźwięku w tym formacie (bitrate *20 ms). W przykładzie przedstawionym na listingu 3 wykorzystano drugą z tych możliwości, aby pokazać sposób cyklicznego podawania ramek dźwiękowych w funkcji obsługi przerwania.

Rozpoczęcie odtwarzania uzyskujemy poprzez wywołanie funkcji *adl_audioStreamPlay()*. Przeglądając możliwe parametry tej funkcji, zauważamy, że oprócz możliwości odtwarzania formatu AMR/AMR-WB jest też możliwość odtwarzania dźwięku w formacie PCM 8-bitowego z próbkowaniem 8 kHz lub 16 kHz (tylko na układ lokalny). Sposób odtwarzania dla PCM niewiele się różni od tego dla formatu AMR i został przedstawiony w przykładowej aplikacji dostępnej wraz ze środowiskiem o nazwie „PCM Record and Play". W aplikacji tej zobaczymy również, w jaki sposób rejestrować dźwięk przychodzący z toru GSM lub mikrofonu lokalnego.

Należy powiedzieć jeszcze kilka słów, w jaki sposób zostały przygotowane próbki dźwiękowe zawarte w tablicy rawData[] w pliku „data.h".

Do konwersji dźwięku najlepiej użyć jakiegoś programu przeznaczonego do konwersji multimediów. W moim przypadku był to program dostępny na stronie Nokii – Nokia Multimedia Converter. Konwertujemy dźwięk do odpowiedniego formatu i zapisujemy na dysku.

Po dokonaniu konwersji na format AMR należy użyć opisanego w jednym z poprzednich odcinków programu bin2h.exe. Program ten utworzy nam plik z rozszerzeniem .h, który będzie zawierał dane pliku audio w postaci wektora danych. Pliki z danymi AMR oprócz informacji o dźwięku mają na samym początku nagłówek informujący, czy dane są w formacie AMR, czy AMR-WB. Możemy to sprawdzić, podglądając plik z rozszerzeniem AMR za pomocą dowolnego edytora Hex. Powinniśmy zobaczyć na początku pliku poniższy ciąg znaków:
"#!AMR\n"
dla AMR-WB będzie to wyglądało tak:
"#!AMR-WB\n"

Niestety Open AT nie potrafi samodzielnie pomijać nagłówków, więc powinniśmy usunąć (najlepiej już z pliku .h) pierwszych 6 bajtów dla formatu AMR lub 9 bajtów dla formatu AMR-WB z tablicy zawierającej dane dźwięku stworzonej przez program bin2h. Tak przygotowany plik z rozszerzeniem .h kopiujemy metodą kopiuj wklej do katalogu inc naszego projektu w Project Exploratorze.

Ostatnią zagadnieniem, jakie przedstawimy w tym odcinku, będzie przykładowa aplikacja dekodująca tony DTMF przychodzące z odebranego połączenia GSM (**listing 4**). Jest to dość ciekawa funkcjonalność z punktu widzenia możliwości zdalnego sterowania naszą aplikacją.

Listing 4. Dekoder DTMF

```
#include "adl_global.h"
#include "generated.h"
// audio handle
s32 audioHandle, DTMFHighIrqHandle;
u32 blank_duration = 50;
void * StreamBuffer;   // audio stream buffer
adl_audioPostProcessedDecoder_t * PostProcessedDTMF;
ascii ResultStr [300];
bool DTMFHighIrqHandler( adl_irqID_e Source, adl_irqNotificationLevel_e NotificationLevel, adl_irqEventData_t *Data )
{
  TRACE (( 1, "High IRQ handler" ));
  PostProcessedDTMF =  ( adl_audioPostProcessedDecoder_t * )  ( ( adl_audioStream_t * )Data -> SourceData )->
DataBuffer ;
  wm_sprintf ( ResultStr, "odebrano DTMF : %c ; cz.trwania : %d ms\r\n", PostProcessedDTMF->DecodedDTMF, (
PostProcessedDTMF->Duration * 10 ) );
  adl_atSendResponse( ADL_AT_RSP, ResultStr );
  return FALSE;
}

void MyAudioEventHandler (s32 audioHandle, adl_audioEvents_e Event){
  TRACE((1,"MyAudioEventHandler"));
  return;
}

s8 CallHandler ( u16 Event, u32 Call_ID ){
  TRACE((1,"Event= %d", Event));
  switch (Event){
    case ADL_CALL_EVENT_RING_VOICE:
      adl_callAnswer ();   //odbierz polaczenie
      break;
    case ADL_CALL_EVENT_ANSWER_OK:
      TRACE((1,"Polaczenie odebrane"));
      break;
    case ADL_CALL_EVENT_RELEASE_ID:
      adl_audioStop ( audioHandle );   //zatrzymaj
      adl_audioUnsubscribe ( audioHandle );
      if( StreamBuffer != NULL )
        adl_memRelease( StreamBuffer );  //zwolnij bufor
      break;
    }
  return ADL_CALL_NO_FORWARD;
}

void main_task ( void ){
  s32 sReturn;
  s32 BufferSize;
  TRACE (( 1, „Embedded Application : Main" ));
  adl_callSubscribe(CallHandler);
  DTMFHighIrqHandle = adl_irqSubscribe ( DTMFHighIrqHandler, ADL_IRQ_NOTIFY_HIGH_LEVEL, ADL_IRQ_PRIORITY_HIGH_LEVEL, 1
);
  audioHandle = adl_audioSubscribe ( ADL_AUDIO_VOICE_CALL_RX, MyAudioEventHandler, ADL_AUDIO_RESOURCE_OPTION_FORBID_
PREEMPTION );
  adl_audioSetOption ( audioHandle, ADL_AUDIO_DTMF_DETECT_BLANK_DURATION, blank_duration  );
  adl_audioGetOption ( audioHandle, ADL_AUDIO_DTMF_PROCESSED_STREAM_BUFFER_SIZE, &BufferSize );
  StreamBuffer = adl_memGet( BufferSize );
  sReturn = adl_audioStreamListen ( audioHandle, ADL_AUDIO_DTMF, 0, DTMFHighIrqHandle, StreamBuffer );
  if( sReturn < OK ) TRACE (( 1, „Blad : %d", sReturn ));
    else
  TRACE (( 1, „DTMF : start" ));
}
```

Chociaż w aplikacji z listingu 4 wykorzystano jedynie przerwanie wysokiego poziomu, jednak pamięć stosu należy zadeklarować również i dla poziomu niskiego przerwań (za pomocą formatki *generated.c*).

Działanie rozpoczyna się od subskrypcji do obsługi połączeń GSM. Podajemy handler, który będzie wywołany w momencie, gdy pojawi się jakieś połączenie przychodzące typu VOICE. Takie połączenie zostanie automatycznie odebrane przez aplikację.

Kolejne kroki to subskrypcja do serwisu Audio i wskazanie jako argumentu toru GSM – to stąd będą odbierane nasze sygnały DTMF. Następnie ustawiamy odstęp ciszy (w ms), jaka powinna być między dwoma tonami DTMF, aby zostały one odróżnione oraz rozmiar bufora na znaki. Ostatni krok to rozpoczęcie detekcji za pomocą funkcji *adl_audioStreamListen()*.

Jeśli teraz zadzwonimy do naszego urządzenia, to połączenie zostanie odebrane automatycznie. Naciskając różne cyfry na klawiaturze telefonu komórkowego, spowodujemy wygenerowanie tonów DTMF, które zostaną odebrane i zdekodowane przez modem. Efektem tego będzie informacja wysyłana przez port szeregowy.

Jeżeli rozłączymy nasze połączenie, to zostanie wywołana funkcja *CallHandler()* ze zdarzeniem *ADL_CALL_EVENT_RELEASE_ID*. Spowoduje to zwolnienie serwisu Audio oraz zwolnienie pamięci wcześniej zarezerwowanej na bufor.

Podsumowanie

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.