Domofon bezprzewodowy

Przykład użycia platformy OpenAT

Na łamach Elektroniki Praktycznej wielokrotnie były prezentowane rozwiązania oparte o komunikację GSM. Zazwyczaj opierały się one o dedykowane moduły będące modemami, sterowane komendami Hayesa. Nie inaczej jest w przypadku Q2687 firmy Sierra Wireless (dawniej Wavecom), bohatera tego artykułu. Jest to niewielka płytka z mikrokontrolerem z rdzeniem ARM9, pracująca pod kontrolą systemu operacyjnego OpenAT, który – prócz standardowych usług systemu operacyjnego – dostarcza także funkcje obsługi transmisji GSM. Takie połączenie było strzałem w dziesiątkę firmy Sierra Wireless: aby stworzyć całkiem zaawansowaną aplikację, nie potrzeba mieć specjalistycznej wiedzy z zakresu telekomunikacji – wystarczy podstawowa znajomość języka C oraz API systemu OpenAT.

Kolejną zaletą modułu, znacznie odróżniającą od wspomnianych urządzeń, jest obecność w jego strukturze pokaźnej liczby interfejsów: od zwykłych linii GPIO, przez SPI, I²C, dedykowane złącza do zewnętrznej klawiatury, po porty UART i USB. To zmieniło dotychczasową sytuację, w której moduły GSM do tej pory spełniały rolę jedynie modemów. Moduł Q2687 dzięki wymienionym peryferiom w większości praktycznych zastosowań nie potrzebuje mikrokontrolera zewnętrznego – wszystkie niezbędne elementy są zawarte w jego strukturze.

OpenAT – co to takiego?

OpenAT jest wielozadaniowym systemem operacyjnym czasu rzeczywistego z wywłaszczaniem. Według zapewnień producenta to jedyny system, który natywnie dostarcza w swoim API funkcje obsługi usług bezprzewodowych.

Pisząc programy pod kontrolą można wybrać jedną z dwóch metod. W pierwszej z nich tworzy się programy, których usługi są wywoływane przez funkcje systemu operacyjnego, np. dla ustanowienia połączenia będzie to *adl_callSetup(„697661441”, ADL_CALL_MODE_VOICE)*. Przy użyciu drugiej wywołuje się komendy AT za pomocą funkcji *adl_atCmdSend()* (na przykład *adl_atCmdSend(„ATDT 697661441”, ATIRspHandler)*).

Komendy do modemu można również przesyłać w postaci tekstu poprzez interfejs UART lub USB. Korzystając z tej metody oraz z tego, że prawie wszystkie funkcje zawarte w API mają swoje odpowiedniki AT, można tworzyć aplikacje nawet bez znajomości API, a jedynie znając nieco rozszerzone komendy AT Hayes'a. Oczywiście, trudno w takiej sytuacji mówić o programowaniu z użyciem API. Warto również wspomnieć, że korzystając z OpenAT można również tworzyć własne komendy AT.

Drugą charakterystyczną cechą programów pracujących pod kontrolą systemu OpenAT jest ich stopień skomplikowania, zależny od szczegółowości zasięgu kontroli poszczególnych elementów. Pod tym stwierdzeniem należy rozumieć monitorowanie lub pomijanie zdarzeń związanych np. z połączeniem głosowym. Jeśli chcemy kontrolować takie zdarzenia, należy wywołać funkcję z rodziny *adl_XXXSubscribe-(wskaźnik_na_funkcję)*, gdzie jest *XXX* zależne od rodzaju obsługiwanych zdarzeń. Wtedy, za każdym razem, gdy wystąpi zdarzenie związane z połączeniem, będzie wywoływana funkcja o adresie *wskaźnik_na_funkcję*.

System dostarcza mechanizmów, takich jak: timery, obsługa pamięci (o dostępie swobodnym RAM i wbudowanej pamięci FLASH), semafory, przerwania (wewnętrzne i zewnętrzne) oraz inne, charakterystyczne dla systemów operacyjnych. Ale – jak to było wspomniane wcześniej – dostarcza także API do obsługi usług sieci GSM oraz urządzeń peryferyjnych. Obok „zwyczajnych” funkcji obsługi np. połączeń głosowych czy wysyłania i odbioru komunikatów SMS, zaimplementowano również funkcje specjalne, jak szyfrowanie – warto przyjrzeć się dokumentacji API.

Na stronie producenta (*www.sierrawireless.com*) znajduje się darmowe środowisko IDE oparte na popularnym Eclipse. Zawiera ono edytor, kompilator, debugger oraz terminal do komunikacji z Q2687, czyli wszystko co jest niezbędne do pracy z modułem. Program może zaczynać się w dwojaki sposób: albo wywołując funkcję *adl_main()*, w której wykonywane są kolejne jego kroki, albo umieszczając zadania w specjalnej strukturze *adl_InitTasks_t*. Jeśli zostanie ona zdefiniowana, zadania są wywoływane automatycznie.

Przykładowy projekt – domofon

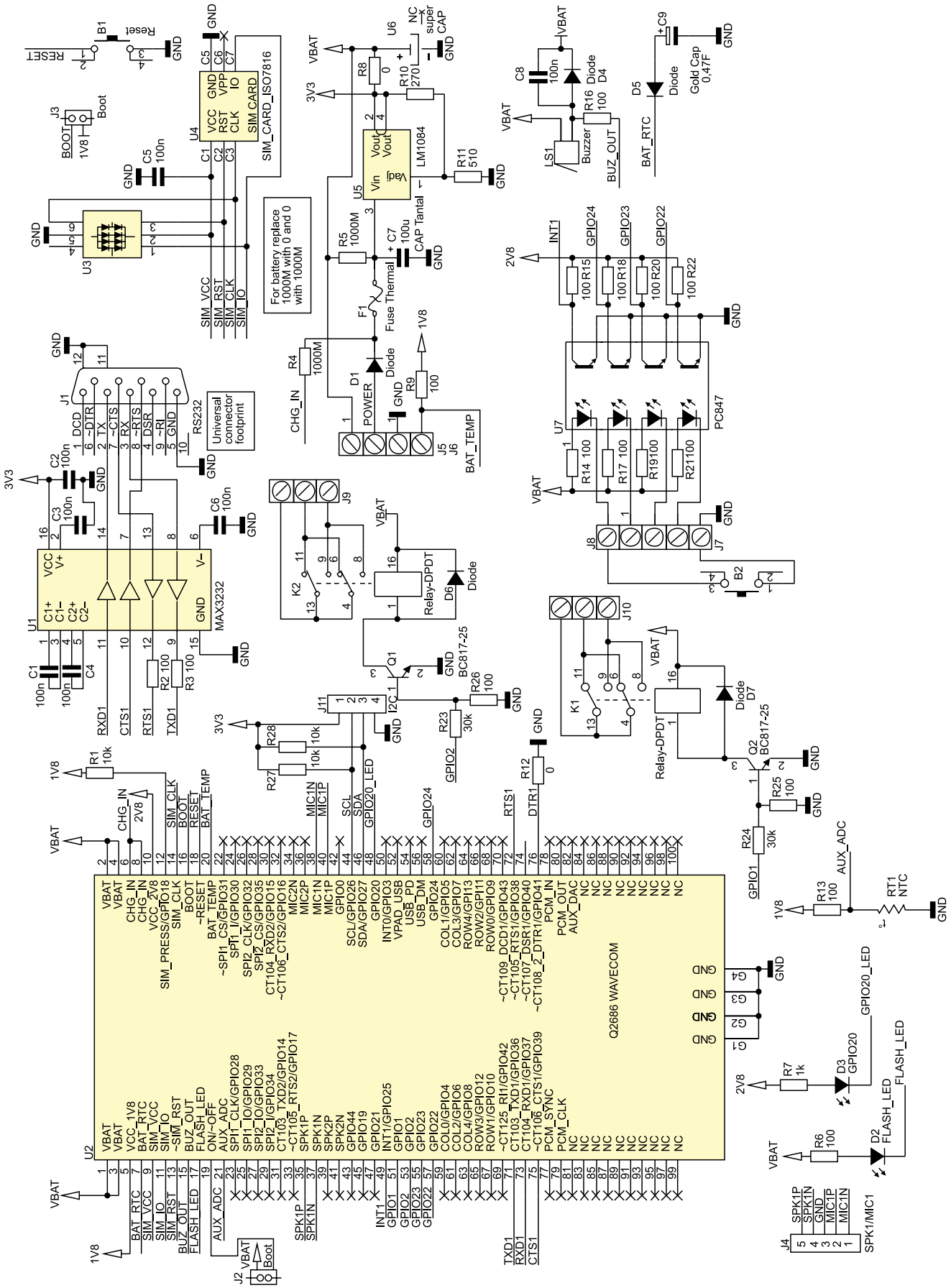
Działanie modułu zostanie zademonstrowane na przykładzie domofonu bezprzewodowego. Chodzi o to, aby odstraszyć potencjalnego złodzieja np. przed rabunkiem mieszkania. W momencie, gdy osoba stojąca przed domem wciśnie przycisk domofonu, wykonywane jest połączenie telefoniczne do właściciela, który może być – przykładowo – za granicą, natomiast osoba stojąca przed domofonem nic o tym nie wie. Dodatkowo, właściciel za pomocą klawiatury swojego telefonu może sterować urządzeniami podłączonymi do przekaźników. Pierwsze wciśnięcie danej cyfry powoduje włączenie, następne – wyłączenie.

Budowa proponowanego urządzenia jest pokazana na **rysunku 1**. Przycisk B2 służy do wyzwalania zewnętrznego przerwania, który z kolei powoduje wykonanie połączenia głosowego na z góry zdefiniowany numer telefonu (właściciela). Za pomocą przycisków telefonu można sterować przekaźnikami K1 oraz K2.

Kod programu zamieszczono na **listingu 1**. Działanie zaczyna się w funkcji *adl_main()*. Na początek zostaje wywołana funkcja tworząca odbiornik zdarzeń zgłaszanych od karty SIM. Jej pierwszy argument to numer PIN karty, natomiast drugi to nazwa funkcji wywoływanej przy zdarzeniu. Jeśli teraz spojrzymy na kod funkcji *SimHandler()*, to zobaczymy, że przyjmuje ona jeden parametr – *Event*. W zależności od zdarzenia, przybiera ona inną wartość (przedstawione są one w możliwych wartościach instrukcji *switch*). Nas interesuje *ADL_SIM_EVENT_FULL_INIT*, co oznacza, że z powodzeniem przeprowadzono pełną inicjalizację. W tym momencie wywoływana jest funkcja pomocnicza inicjalizująca działanie reszty modułu. Gdybyśmy od razu przeprowadzili tę inicjalizację, bez czekania na zakończenie *ADL_SIM_EVENT_FULL_INIT*, to bardzo możliwe, że nie wszystkie peryferia byłyby gotowe do

obsługi na czas (np. niemożliwe by było wykonanie połączenia głosowego, ponieważ nie został jeszcze sprawdzony kod PIN).

Pierwsza linia funkcji *init()* konfiguruje piny zewnętrzne zgodnie z parametrami zawartymi w tablicy struktur *gpioOutConfig* typu *adl_ioDefs_t*. Ustawiają one tryb pracy pinu (wejściowy lub wyjściowy) oraz poziom napięcia. Następna linia kodu odpowiada za utworzenie odbiornika zdarzeń od przerwania, natomiast kolejna konfiguruje



Rysunek 1. Schemat ideowy domofonu GSM

Listing 1. Oprogramowanie domofonu pracujące pod kontrolą OpenAT

```
#include „adl_global.h”
#include „generated.h”

#define PIN_CODE NULL //pin w formie „xxxx”, lub NULL gdy brak
#define NR_TEL „509220049”

s8 isCalling=0;
const u32 adl_InitIRQLowLevelStackSize = 1024; //potrzebne dla adl_irqSubscribe
const u32 adl_InitIRQHighLevelStackSize = 1024;

void init(void);
s32 gpioHandler;
adl_ioDefs_t* gpioOutConfig[2] ={
                {ADL_IO_GPIO | 1 | ADL_IO_DIR_OUT | ADL_IO_LEV_LOW},  //GPIO1 - pin 51
                {ADL_IO_GPIO | 2 | ADL_IO_DIR_OUT | ADL_IO_LEV_LOW},  //GPIO2 - pin 53
                };
adl_ioDefs_t gpio1= ADL_IO_GPIO | 1;
adl_ioDefs_t gpio2= ADL_IO_GPIO | 2;
u32 zmienna=1;

s32 ExtIntHandler;
s32 IrqHandler;
adl_extintConfig_t extintConfig={ADL_EXTINT_SENSITIVITY_FALLING_EDGE, ADL_EXTINT_FILTER_DEBOUNCE_MODE, 4, 0, NULL};
#define EXTINT_PIN1 1  //INT1 - pin 49 (GPIO25), 2V8

s32 IRQHandler;
s32 audioHandler;
adl_audioPostProcessedDecoder_t *StreamBuffer;

//------------------------------------------------------------------------------
s8 callHandle(u16 Event, u32 Call_ID){
        TRACE((1, „Event: %i, ID: %i:)”, Event, Call_ID));
        if(Event==ADL_CALL_EVENT_NO_CARRIER){
                adl_callHangup();
                isCalling=0;
                TRACE((1, „HUNGUP no carrier”));
        }
        if(Event==ADL_CALL_EVENT_RING_VOICE){
                adl_callAnswer();
                isCalling=1;
                TRACE((1, „RING”));
        }
}
//------------------------------------------------------------------------------
void SimHandler(u8 Event){
        switch(Event){
        case ADL_SIM_EVENT_PIN_OK:
```

Listing 1. c.d.

```
                TRACE (( 1, „SimHandler: ADL_SIM_EVENT_PIN_OK” ));
                break;
        case ADL_SIM_EVENT_REMOVED:
                TRACE (( 1,  „\r\nSimHandler: ADL_SIM_EVENT_REMOVED” ));
                break;
        case ADL_SIM_EVENT_INSERTED:
                TRACE (( 1,  „SimHandler: ADL_SIM_EVENT_INSERTED” ));
                break;
        case ADL_SIM_EVENT_FULL_INIT:
                TRACE (( 1,  „SimHandler: ADL_SIM_EVENT_FULL_INIT” ));
                init();
                break;
        default:
                TRACE (( 1, „SimHandler Event: %i”, Event));
        }
}
//------------------------------------------------------------------------------
//funkcja obslugi przerwania od zewnetrznego przycisku
bool INTHandler(adl_irqID_e Source, adl_irqNotificationLevel_e NotificationLevel, adl_irqEventData_t *Data){
        TRACE (( 1, „Przerwanie” ));
        if(isCalling==0){
                adl_callSetup(NR_TEL, ADL_CALL_MODE_VOICE);
                isCalling=1;
        }
        return TRUE;
}
//------------------------------------------------------------------------------
bool MyLowLevelIRQHandler(adl_irqID_e Source, adl_irqNotificationLevel_e NotificationLevel, adl_irqEventData_t *Data){
                TRACE((1, „DTMF recived: %c”, StreamBuffer->DecodedDTMF ));
                if(StreamBuffer->DecodedDTMF == ,1’){
                        TRACE((1, „ZMIANA 1”));
                        adl_ioWriteSingle(gpioHandler, &gpio1, 1-adl_ioReadSingle(gpioHandler,
                                &gpio1));
                }
                if(StreamBuffer->DecodedDTMF == ,2’){
                        TRACE((1, „ZMIANA 2”));
                        adl_ioWriteSingle(gpioHandler, &gpio2, 1-adl_ioReadSingle(gpioHandler,
                                &gpio2));
                }
        return FALSE;
}
//------------------------------------------------------------------------------
void MyAudioEventHandler(s32 audioHandle, adl_audioEvents_e Event){
        TRACE (( 1, „audioSubscribe” ));
        return;
}
//------------------------------------------------------------------------------
void main_task (void){
        //ustawienie obslugi polaczen
        adl_simSubscribe(SimHandler, PIN_CODE);

        //ustawienie funkcji zglaszajacej zdarzenia polaczenia
        adl_callSubscribe(callHandle);
}
//------------------------------------------------------------------------------
void init(void){
        //ustawienie przyciskow gpio jako wyjscia
        gpioHandler=adl_ioSubscribe(2, gpioOutConfig, 0, 0, 0);

        //ustawienie przerwan od zewnetrznego przycisku
        IrqHandler  = adl_irqSubscribe(INTHandler, ADL_IRQ_NOTIFY_LOW_LEVEL,
                                ADL_IRQ_PRIORITY_HIGH_LEVEL, ADL_IRQ_OPTION_AUTO_READ );
        ExtIntHandler=adl_extintSubscribe(EXTINT_PIN1, 0, IrqHandler, &extintConfig);

        //DTMF
        s32 BufferSize;
        IRQHandler=adl_irqSubscribe(MyLowLevelIRQHandler, ADL_IRQ_NOTIFY_LOW_LEVEL,
                                ADL_IRQ_PRIORITY_LOW_LEVEL, ADL_IRQ_OPTION_AUTO_READ );
        audioHandler=adl_audioSubscribe(ADL_AUDIO_VOICE_CALL_RX, MyAudioEventHandler,
                                 ADL_AUDIO_RESOURCE_OPTION_FORBID_PREEMPTION);
        adl_audioGetOption(audioHandler, ADL_AUDIO_DTMF_PROCESSED_STREAM_BUFFER_SIZE,
                        &BufferSize);
        StreamBuffer=adl_memGet(BufferSize);
        adl_audioSetOption(audioHandler, ADL_AUDIO_DTMF_DETECT_BLANK_DURATION,
                        ADL_AUDIO_RAW_DTMF_SAMPLE_DURATION * ADL_AUDIO_MAX_DTMF_PER_FRAME * 6);
        adl_audioStreamListen(audioHandler, ADL_AUDIO_DTMF, IRQHandler, 0, StreamBuffer);
}
```

to przerwanie jako przerwanie zewnętrzne. Struktura *extintConfig* typu *adl_extintConfig_t* określa, czy przerwanie jest wyzwalane zboczem, czy poziomem sygnału oraz czy włączyć eliminowanie drgań styków (debouncer). W momencie wystąpienia zewnętrznego przerwania (wciśnięto przycisk B2), jest wywoływana funkcja *INTHandler()*, która z kolei wykonuje połączenie telefoniczne, w zależności od wartości zmiennej *isCalling*. Jest ona stosowana, aby zabezpieczyć wywołanie wykonania połączenia w czasie, gdy takie jest już ustanowione. Makrodefinicja *TRACE()* jest wykorzystywana tylko przy debugowaniu i nie wpływa na działanie układu.

Po skonfigurowaniu przerwania zewnętrznego, ustanawiana jest możliwość sterowania za pomocą DTMF (klawisze telefonu). Tu sprawa jest trochę bardziej skomplikowana, niż to było do tej pory. Do ustawienia dekodowania sygnałów DTMF z kanału odbiorczego (RX) połączenia głosowego służy funkcja *adl_audioStreamListen()*. Aby ją można było wywołać, należy wcześniej utworzyć odbiornik zdarzeń od przerwania (tak jak poprzednio) oraz odbiornik zdarzeń audio. Dodatkowo, należy uzyskać rozmiar bufora, w którym będą zdekodowane kody oraz przyporządkować mu obszar pamięci. Po tych czynnościach, za każdym razem, gdy podczas rozmowy głosowej zostanie wciśnięty klawisz klawiatury numerycznej, będzie wywoływana funkcja *MyLowLevelIRQHandler()*, a w strukturze *StreamBuffer* w polu *DecodedDTMF* zostanie zapamiętany kod wciśniętego przycisku. Przez analogię, można dodawać wywołania zdarzeń dla innych klawiszy.

W *adl_main()* zdefiniowano funkcję obsługi zdarzeń od połączeń. Pierwsza instrukcja *if* odbiera zdarzenie odłożenia przez rozmówcę (właściciela) słuchawki. Jeśli takie

zachowanie wystąpi, moduł zachowuje również rozłącza się. Drugi blok *if* jest prawdziwy, gdy występuje próba nawiązania połączenia głosowego z zewnątrz – wtedy moduł odbiera słuchawkę. Takie zachowanie zostało zaimplementowane, aby właściciel mógł w dowolnym momencie wykonać połączenie i sterować przekaźnikami.

Na koniec należy zaznaczyć, że aby było możliwe korzystanie z obsługi zdarzeń od przerwania (funkcja *adl_irqSubscribe()*), należy pamiętać o zdefiniowaniu dwóch stałych *adl_InitIRQLowLevelStackSize* oraz *adl_InitIRQHighLevelStackSize* odpowiedzialnych za wielkość stosu. Bez tego funkcja *adl_irqSubscribe()* zwróci błąd.

Schemat urządzenia

Rysunek 1 przedstawia schemat proponowanego rozwiązania. Jego sercem jest modem Q2687. W jego aplikacji użyto tylko części interfejsów. Niektóre z elementów umieszczonych na schemacie znalazły się na nim, ponieważ (np. termistor NTC dołączony do wejścia przetwornika A/C, nóżka 21) początkowo płytka została zaprojektowana jako ewaluacyjna. Niemniej jednak, nie przeszkadza to w działaniu, bardziej dociekliwym Czytelnikom daje pole do dalszych eksperymentów.

Płytkę wyposażono w złącze RS-232 z konwerterem napięć MAX232 (układ U1). Interfejs RS-232 służy do programowania oraz komunikacji poprzez UART (115200 baudrate, 8 bitów danych, brak bitu parzystości, jeden bit stopu), natomiast zwora J3 – dołączona do pinu oznaczonego jako BOOT – do wprowadzenia modułu w alternatywny tryb wgrywania oprogramowania firmowego (firmware) w wypadku, gdy nie ma możliwości wykorzystania standardu XMODEM. Wtedy potrzebna jest odpowiednia aplikacja dostarczana przez Sierra Wireless.

Sygnały wejścia/wyjścia są wyprowadzone na złącze J8. Za ego pomocą dołączono przycisk B2 służący do wyzwalania przerwania („dzwonienia" domofonem). Całość jest odseparowana galwanicznie od Q2687 przez transoptor (układ U7), zabezpieczając płytkę przed uszkodzeniem. Ponieważ wydajność prądowa linii GPIO modemu jest nie większa niż 2 mA, do sterowania przekaźnikami K1 i K2 potrzebne są tranzystory.

Słowo wyjaśnienia należy się także układowi zasilania. Z tego względu, że modem jest przystosowany do działania z zasilaniem bateryjnym, napięcie nie powinno przekraczać 4 V, natomiast maksymalna moc pobierana ze źródła zasilania jest podczas nadawania nie większa niż 4 W. Dlatego prąd wejściowy może sięgać 1 A (w impulsie). Dla złagodzenia skutków udaru prądowego, zastosowano kondensator XCap o pojemności 0,4 F.

Rolę anteny nadawczo/odbiorczej w modelu pełni odcinek zwykłego przewodu. W naszym wypadku jest on wystarczający. Należy tylko pamiętać, aby jego długość była odpowiednia – 1/4 długości fali dla częstotliwości 1,9 GHz to 4 cm.

Podsumowanie

Celem autorów było przekonanie Czytelników, że stworzenie urządzenia komunikacyjnego opartego o komunikację GSM nie musi być trudne. Źródło programu zajmuje nieco powyżej 100 linii, a dodatkowe funkcjonalności można zaimplementować poprzez zdefiniowanie funkcji obsługi zdarzeń, a w nich operowanie na specyficznych (dla danej funkcjonalności) danych. Zajmuje to kilka lub kilkanaście dodatkowych linii kodu, co nie jest dużo. Przykładowo, łatwo sobie wyobrazić skonstruowanie kompletnego telefonu komórkowego za pomocą systemu OpenAT, a ile trzeba by było wiedzy i umiejętności, aby zbudować samemu go od podstaw? Odpowiedź jest chyba oczywista.

Na koniec chciałbym podziękować dr inż. Jackowi Majewskiemu za podsunięcie pomysłu, pomoc merytoryczną oraz wypożyczenie sprzętu.

Łukasz Tafelski
lukasz.tafelski@gmail.com