SPPoB – prosty protokół pakietowy dla automatyki inteligentnego budynku (2)

Protokół SPPoB (Simple Packet Protocol over Bus) opisany w poprzedniej części artykułu został z powodzeniem zaimplementowany i przetestowany na trzech różnych platformach sprzętowych. W tej części opisuję przykładową organizację plików i katalogów dla protokołu SPPoB, a także prezentuję oprogramowanie realizujące komunikację SPPoB dla mikrokontrolerów rodziny AVR, STM32 i AT91SAM. W materiałach do artykułu dostępne są trzy przykładowe projekty, które można wykorzystać zarówno bezpośrednio, jako projekty startowe do budowanych urządzeń, lub też użyć ich jako referencji dla własnej implementacji. Projekty zostały opracowane dla kompilatorów GCC działających w środowisku Windows: avr-gcc (WinAVR) oraz arm-none-eabi-gcc (m.in. z pakietu Yagarto).

Jako płytki bazowe dla pierwszych eksperymentów można z powodzeniem wykorzystać płytki ewaluacyjne lub tzw. „znaleziska z szuflady". Wymagania sprzętowe są bowiem niewielkie i sprowadzają się głównie do prawidłowego podłączenia zasilania mikrokontrolera oraz zapewnienia interfejsu RS-485 na jednym z U(S)ART-ów (np. wg schematów ogólnych zamieszczonych w pierwszej części artykułu lub bardziej szczegółowych, przedstawionych poniżej). Może przydać się także dioda LED sygnalizująca w każdym urządzeniu aktywność sieciową.

Kompletne „projekty demo" dołączone do artykułu zostały przetestowane na mikrokontrolerach: ATmega8L, STM32F103RET6 i AT91SAM7XC256. Podłączenie niezbędnych układów peryferyjnych jest podobne lub identyczne dla różnych wariantów wymienionych wyżej mikrokontrolerów. Dzięki temu przedstawione dalej schematy można traktować jako referencyjne dla kompatybilnych modeli mikrokontrolerów, np. podłączenie ATmega88 będzie identyczne jak ATmega8A. Podobnie, podłączenie STM32F4x będzie od zewnątrz takie samo jak STM32F1x. Natomiast przenoszenie programu (napisanie „portu" obsługi protokołu) może wymagać różnego nakładu pracy, zależenie od tego, jak bardzo mikrokontrolery różnią się między sobą (np. port dla ATmega88 wymaga minimalnego wysiłku, niewiele trudniej jest przenieść obsługę SPPoB z SAM7X na SAM7S lub SAM9).

Sposób dołączenia transceivera do mikrokontrolera ATmega8 przedstawiono na **rysunku 1**. Przykładowy program z katalogu *EP_ATmega8_Demo* działa z taką właśnie konfiguracją sprzętową oraz aktywnym zewnętrznym oscylatorem kwarcowym o częstotliwości 7,3728 MHz (należy pamiętać o ustawieniu „fusebitów").

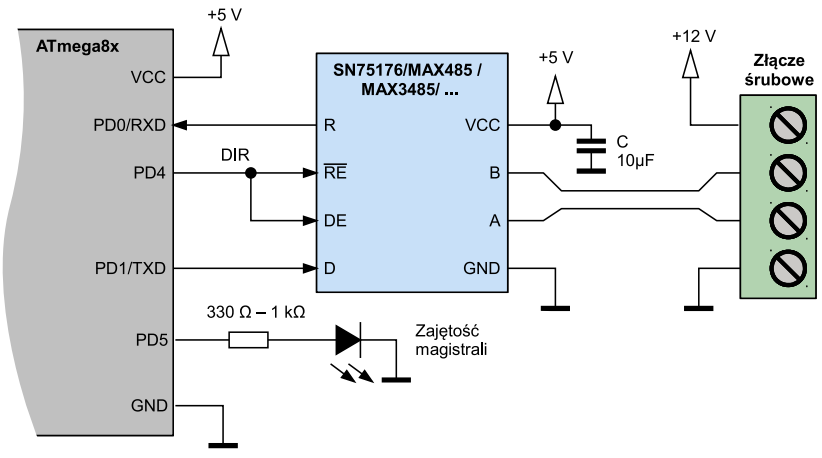
Przykładowy port i program demonstracyjny SPPoB dla mikrokontrolerów STM32F103 znajdziemy w katalogu *EP_STM32F103_Demo*, natomiast na **rysunku 2** przedstawiono schemat podłączeń warstwy sprzętowej. Mikrokontroler powinien pracować z oscylatorem kwarcowym o częstotliwości 8 MHz (jest to typowa wartość dla wielu projektów na STM32F1x). Wyprowadzenie BOOT0 mikrokontrolera podłączamy do masy, aby po zerowaniu startował program z jego głównej pamięci Flash.

I wreszcie port dla mikrokontrolera AT91SAM7XC256 znajdziemy w katalogu *EP_SAM7_Demo*, a sposób podłączenia warstwy fizycznej SPPoB na **rysunku 3**. Do uruchomienia aplikacji testowej, mikrokontroler powinien pracować z wewnętrznym sygnałem zegarowym o częstotliwości 48 MHz, co najczęściej uzyskuje się używając oscylatora kwarcowego 18,432 MHz i standardowego filtru PLL spotykanego na większości płytek ewaluacyjnych.

Organizacja katalogów i plików

Projekty testowe oraz pliki .c i .h do obsługi protokołu SPPoB znajdują się na jednym poziomie, w jednym, głównym katalogu (**rysunek 4**). Każdy projekt ma swój podkatalog, a w nim własne moduły z głównym programem, Makefile, sterowniki układów peryferyjnych i wszystkie inne pliki potrzebne do poprawnej kompilacji i działania.

Obsługa SPPoB jest zaimplementowana w module *spp.c* oraz – częściowo – w plikach nagłówkowych specyficznych dla platformy. Kod został podzielony tak, aby fragmenty potrzebne dla wszystkich platform znalazły się w module *spp.c*, natomiast funkcje specyficzne dla danych platform w odpowiednich plikach nagłówkowych, tutaj *spp_m8.h*, *spp_sam.h* i *spp_stm32.h*.



Rysunek 1. Podłączenie układów peryferyjnych w projekcie przykładowym dla mikrokontrolerów ATmega8/8A/88 i kompatybilnych

Plik nagłówkowy *spp.h* należy dołączyć przy pomocy dyrektywy *#include* do modułu, w którym chcemy korzystać z protokołu SPPoB. Zawiera on prototypy wszystkich funkcji interfejsowych (tzn. nie „ukrytych" przed użytkownikiem) oraz definicje typów i przydatne stałe w postaci makrodefinicji.

Plik nagłówkowy *spp_cmdid.h* jest trochę „nadobowiązkowy": jego treść jest dołączana do pliku *spp.h* i zawiera on makrodefinicje z komendami i identyfikatorami, które mogą być przydatne dla rozróżniania przeznaczenia pakietów (czy pakiet zawiera polecenie, dane, zapytanie – wg **tabeli 1** zamieszczonej w pierwszej części artykułu).

Dane konfiguracyjne specyficzne dla konkretnego projektu można umieszczać w pliku *spp_cfg.h*. Możemy tam skonfigurować przede wszystkim, jaki jest adres urządzenia, a także zmodyfikować kilka innych parametrów – mowa o tym w dalszej części artykułu.

Z racji, że korzystamy z modułu *spp.c* wspólnego dla różnych platform sprzętowych, czasem mogą pojawić się „dziwne błędy" kompilatora, a konkretnie linkera. Biorą się one stąd, że linker będzie próbował wykorzystać już uprzednio utworzone pliki wynikowe i, jeśli zostały one wygenerowane przez inny kompilator lub w inaczej skonfigurowanym projekcie, to pojawią się błędy. Dlatego, przechodząc pomiędzy projektami dla SPPoB, warto najpierw wydać polecenie ***make clean***, które usunie pliki powstałe w wyniku poprzednich kompilacji, a dopiero później właściwe polecenie kompilacji, ***make all***.

Obsługa SPPoB we własnym programie

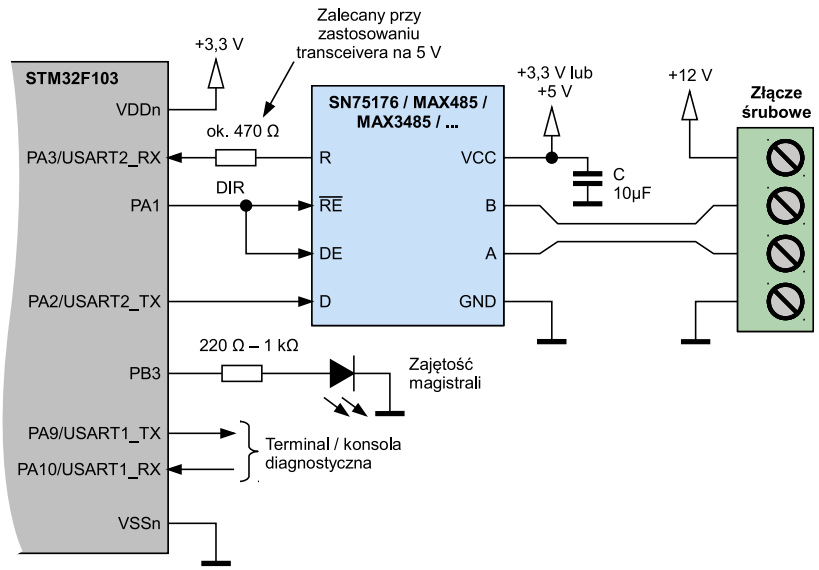
Dostępny dla użytkownika interfejs sterownika SPPoB jest dość prosty. Jeśli chcemy zacząć używać SPPoB w programie, musimy przede wszystkim zainicjalizować sterownik wywołując funkcję ***sspInit*** bez parametrów i bez zwracanej wartości – ta funkcja ma po prostu działać lub wewnętrznie radzić sobie z błędami. Reszta odbywa się w głównej pętli programowej lub w zadaniu bądź procesie systemu operacyjnego (oczywiście jeśli go używamy).

Gdy chcemy korzystać z programowego wykrywania fałszywej zajętości magistrali (timeout dla stanu zajętości), to należy w pętli głównej co pewien czas wywoływać funkcję ***sppIdleTimeout*** bez parametrów i bez zwracanej wartości. To zapewni „odliczanie" czasu zajętości i ewentualne zresetowanie maszyny stanów odbiornika do domyślnego stanu bezczynności (*SPP_STATE_IDLE* – definicja w *spp.c*). Ilość wywołań *sppIdleTimeout* potrzebną do podjęcia decyzji o fałszywej zajętości możemy sobie zdefiniować przy pomocy makra *SPP_IDLE_TIMEOUT* w *spp_cfg.h*. Tym sposobem, jeśli chcemy mieć timeout wynoszący 100 ms, a czas „obrotu" głównej pętli został ustawiony na 1 ms, to do *SPP_IDLE_TIMEOUT* wpiszemy liczbę 100.

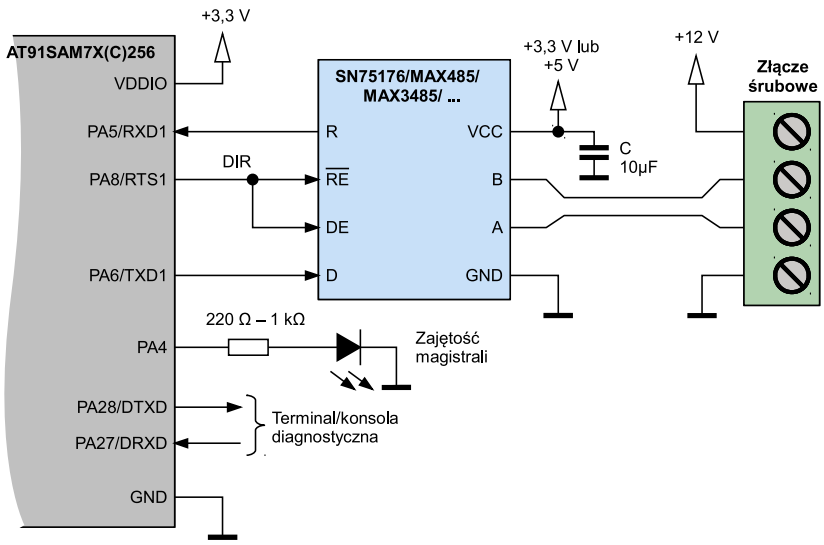
Odbiór pakietów zaimplementowany jest oczywiście w funkcji obsługi przerwania i dzieje się w tle głównego programu. Dopiero, gdy cały pakiet zostanie poprawnie odebrany (zgodzi się adres i suma kontrolna), zostanie przekazana odpowiednia informacja do pętli głównej. Dlatego z każdym „obrotem" głównej pętli programowej lub zadania warto sprawdzić, czy nie odebrano nowego pakietu. Do tego służy funkcja ***sppRx***. Jako parametr należy przekazać do niej wskaźnik do struktury danych typu *T_sppPacket* reprezentującej pakiet. Funkcja zwróci wartość niezerową, gdy w strukturze, do której podaliśmy wskaźnik, czeka na nas komplety, świeżo odebrany pakiet. Jeśli natomiast nic nie przyszło, funkcja zwróci liczbę zero. Funkcję *sppRx* należy wywoływać jak najczęściej, aby uniknąć nadpisania nowymi informacjami pakietu znajdującego się w tymczasowym buforze.

Funkcja ***sppTx*** służy do wysyłania pakietu. W parametrze podajemy wskaźnik do struktury danych typu *T_sppPacket*. Zostaną wysłane takie dane, jakie przygotujemy w strukturze, do której wskaźnik przekazujemy w parametrze. Przed wysłaniem koniecznie trzeba pamiętać o uzupełnieniu pól: adresu docelowego (*dstAddr*), długości danych (*len*) oraz właściwej treści pakietu (pola od *payload[0]* do *payload[len-1]*). Pole adresu źródłowego ustawi się przed wysłaniem automatycznie liczbą zapisaną w makrodefinicji *SPP_ADDRESS* w pliku *spp_cfg.h*.

W założeniach funkcja *sppTx* ma zwracać wartość niezerową, jeśli pakiet zostanie wysłany poprawnie. W prostych implementacjach funkcja nie posiada możliwości „zrezygnowania" z wysłania pakietu nawet, jeśli magistrala będzie zajęta, pakiet zostanie wcześniej czy później wysłany (tu odsyłam Czytelnika do algorytmu sprawdzania zajętości magistrali opisanego w pierwszej części artykułu). Dlatego wartość zwracana będzie zawsze niezerowa. Może się ona jednak przydać, gdy we własnym porcie



Rysunek 2. Dołączenie układów peryferyjnych w projekcie przykładowym dla mikrokontrolera STM32F103 (RS-485 na USART2)



Rysunek 3. Podłączenie układów peryferyjnych w projekcie przykładowym dla mikrokontrolera AT91SAM7X256 lub AT91SAM7XC256 (RS-485 na USART1)

zechcemy uwzględnić bardziej zaawansowane mechanizmy, np. kolejkowania lub sprawdzania poprawności wysyłanych danych.

Właściwie opisane powyżej funkcje wystarczą większości użytkowników, ale można jeszcze skorzystać z paru funkcji dodatkowych. ***sppIsIdle*** pozwala sprawdzić, czy magistrala jest aktualnie wolna (zwracana wartość niezerowa oznacza wolną magistralę). Jest także „automatyczne" odsyłanie potwierdzeń. Jeśli inne urządzenie, które wysłało nam pakiet chce jego potwierdzenia, to wówczas wystarczy wskaźnik do właśnie odebranego pakietu przekazać jako parametr do funkcji ***sppTxAck***. Funkcja automatycznie ustawi adres docelowy na podstawie źródłowego, skróci pole *payload* do rozmiaru 1 bajta i do pola *payload[0]* wpisze liczbę *0* zadaną makrodefinicją *SPP_ID_ACK* (z *spp_cmdid.h*). Funkcja *sppTxAck* zwróci wartość 0 w razie błędu lub wartość niezerową jeśli wysyłanie przebiegło pomyślnie. Tutaj wartości 0 może się pojawić, gdy spróbujemy wywołać tę funkcję dla pakietu, który miał ustawiony adres rozgłoszeniowy do wszystkich urządzeń w sieci lub do wszystkich urządzeń danej klasy (pakiet przyszedł z *dstAddr* ustawionym na *0xFF* lub w jego bitach 3:0 była wartość *0xF*).

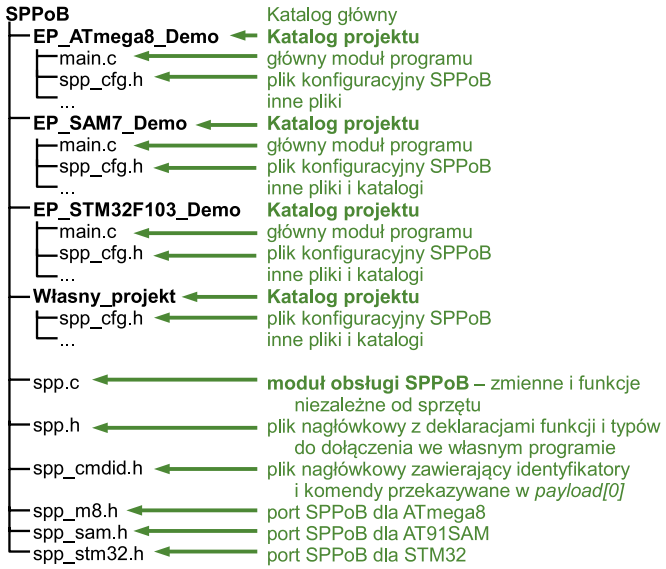
Ustawienia parametrów sterownika SPPoB

W protokole SPPoB przewidziano ustawianie jedynie bardzo podstawowych parametrów. Jak wspomniałem wcześniej, ustawienia zapisane są w plikach *spp_cfg.h* i mogą być różne dla każdego projektu, w szczególności możemy zmieniać adres urządzenia w makrodefinicji ***SPP_ADDRESS***.

Kolejny parametr to ***SPP_PAYLOAD_LEN*** – rozmiar obszaru danych (*payload*) w strukturze typu *T_sppPacket*. Domyślnie we wszystkich projektach przykładowych jest on ustawiony na 40, jeżeli jednak potrzebujemy inną, maksymalnie obsługiwaną długość pakietu, to można go zmieniać w zakresie do 255 (tyle bowiem można zakodować w polu *len*). Może nasunąć się pytanie, co będzie, gdy rozmiar odebranego pakietu przekroczy deklarowany rozmiar bufora? Otóż sterownik został tak napisany, że odbiór pakietów większych niż obsługiwane jest po prostu ignorowany. Wewnętrzna maszyna stanów odbiornika śledzi odbiór wszystkich danych, jednak zapisuje do bufora tylko tyle, ile się zmieści. Pakiety przekraczające rozmiar bufora zostają odrzucone i nie zauważymy ich w wywołaniu funkcji *sppRx*.

Prezentowana implementacja ma możliwość „podwójnego buforowania" włączanego makrodefinicją ***SPP_DOUBLE_RX_BUFFER***. Przy wyłączonym podwójnym buforowaniu pakiet odebrany czeka do momentu, aż odczytamy go w pętli głównej. Wtedy także wyłączone jest przerwanie generowane zakończeniem odbierania znaku, czyli urządzenie nie odbierze innych pakietów. Przy włączonym podwójnym buforowaniu, zawartość pakietu kopiowana jest do tymczasowego bufora jeszcze w funkcji obsługi przerwania, a bufor tymczasowy będzie odczytany w *sppRx*. Tym sposobem bufor odbiorczy jest od razu gotowy na nowy pakiet, a dane oczekujące na odczyt z pętli głównej znajdują się w innym buforze. To jednak kosztuje zajęcie w pamięci operacyjnej miejsca na dwie struktury *T_sppPacket* zamiast jednej.

Do wstępnych testów przydatne może okazać się wyłączenie sprawdzania sum kontrolnych, szczególnie, jeśli mamy zamiar generować pakiety „ręcznie", np. przy pomocy programu terminalowego. Sterownik SSPoB



Rysunek 4. Organizacja katalogów i plików dla kodu źródłowego sterowników i aplikacji SPPoB

Listing 1. Sposób wyboru portu protokołu SPPoB w pliku *spp.c*. Dodając nowe porty należy ten fragment odpowiednio zmodyfikować

```
#if (defined (__AVR_ATmega88__) || \
    defined (__AVR_ATmega8__))
    #include "spp_m8.h"

#elif (defined(AT91SAM7X128) || \
    defined (AT91SAM7X256) || \
    defined (AT91SAM7XC256) || \
    defined (AT91SAM7XC128))
    #include "spp_sam.h"

#elif (defined(STM32F10X_HD) || \
    defined(STM32F10X_MD) || \
    defined(STM32F10X_VL))
    #include "spp_stm32.h"

#else
    #warning Undefined target platform for SPPoB.
#endif
```

Listing 2. Program demonstracyjny dla mikrokontrolera ATmega8

```
#include "../spp.h"
T_sppPacket packet;
void demo(void);

void init(void)
{
    sppInit();
    sei();
}

int main(void)
{
    init();
    while(1)
    {
        sppIdleTimeout();
        demo();
        delay_ms(1);
    }
    return(0);
}

void demo(void)
{
        if(sppRx(&packet))
        {
                delay_ms(SPP_RESPONSE_DELAY_MS);
                packet.dstAddr = SPP_BCAST_ALL;
                sppTx(&packet);
        }
}
```

będzie ignorował niewłaściwe sumy kontrolne, gdy ustawimy ***SPP_IGNORE_CRC*** na wartość niezerową.

Projekt przykładowy dla ATmega8

Program znajdziemy w katalogu ***EP_ATmega8_Demo***. W głównej pętli wywoływana jest funkcja odliczającą *timeout* zajętości oraz funkcja *demo*, w której znajduje zaimplementowany został cały „pokaz możliwości". W głównej pętli dodano także opóźnienie wynoszące ok. 1 ms. W funkcji demo widzimy sprawdzenie, czy został odebrany nowy pakiet. Jeśli tak, to czekamy pewien czas zadany makrodefinicją *SPP_RESPONSE_DELAY_MS* (z pliku *spp.h*), a następnie rozsyłamy treść otrzymanego pakietu do wszystkich urządzeń na magistrali. W tym celu ustawiamy adres docelowy (*dstAddr*) na *SPP_BCAST_ALL* czyli *0xFF*. Na **listingu 2** została przedstawiona główna część opisywanej tutaj aplikacji testowej.

Pakiety „wejściowe" dla projektów przykładowych możemy wygenerować np. przy pomocy programu terminalowego. Doskonale tutaj sprawdza się darmowy program Bray Terminal dostępny pod adresem *http://goo.gl/SmRz3W*. Można w nim utworzyć makra wysyłające dowolne liczby, z których składa się pakiet testowy. W praktyce okazało się to bardzo przydatne przy testowaniu SPPoB.

Oczywiście, potrzebna będzie jakaś prosta przejściówka USB-RS485 mogąca działać w trybie half-duplex. Aby nieco ułatwić sobie zadanie, można sprawdzać działanie systemu z wyłączonym odrzucaniem niewłaściwych sum kontrolnych. W tym celu, w pliku *spp_cfg.h* ustawiamy *SPP_IGNORE_CRC* na 1, a bajt sumy kontrolnej wpisujemy dowolny (koniecznie musimy jakiś wpisać, żeby zgadzała się ilość przesyłanych danych).

Projekt przykładowy dla STM32 i SAM7

Projekty znajdziemy w katalogach ***EP_STM32F103_Demo*** oraz ***EP_SAM7_Demo***. Tutaj funkcje *main* wyglądają bardzo podobnie jak w programie dla ATmega8, jednak zostały bardziej rozbudowane procedury inicjalizacji. Sama funkcja *demo* jest także nieco bardziej rozbudowana (**listing 3**) i korzysta z prostego interfejsu „konsolowego". Adresy urządzeń są ustawione na *0x42* (STM32) i *0x43* (SAM7). Do testów najlepiej będzie włączyć dwa terminale: jeden będzie obsługiwał dane przesyłane przez RS-485, a drugiego można użyć do komunikacji „konsolowej" z urządzeniem (parametry transmisji to 115200 baudów, 8 bitów danych, 1 bit stopu, brak bitu parzystości).

Wysłanie z klawiatury do portu konsoli (USART1 dla STM32 lub DBGU dla SAM7) znaku ‚0' lub ‚1' spowoduje wygenerowanie pakietu, którego akurat w sieci testowej autora używa się do przełączania stanu wyjściowego w kanale 0 sterownika oświetlenia o adresie *0x40*. Analogicznie, po wciśnięciu klawisza ‚1' zmieniamy stan w kanale nr 1 tego samego sterownika. Każdy poprawnie odebrany pakiet sygnalizowany będzie odpowiednim komunikatem na terminalu.

SPPoB we własnym projekcie

Dodanie sterownika SPPoB do własnego projektu nie powinno nastręczać większych trudności. Należy przede wszystkim odpowiednio skonfigurować projekt:

- dodać do kompilacji plik *spp.c*,
- w module, gdzie chcemy korzystać z funkcji wysyłających i odbierających pakiety SPPoB, należy dołączyć dyrektywą *#include* plik *spp.h*,
- w Makefile ustawić nazwę modelu lub rodziny używanego mikrokontrolera, aby pojawiła się globalna definicja odpowiadająca tej nazwie (np. tak jak dla WinAVR wpisujemy „MCU = atmega8"),
- utworzyć w swoim projekcie plik *spp_cfg.h* zawierający konfigurację parametrów zależnych od konkretnego urządzenia.

Później należy zadbać o odpowiednio częste wywoływanie funkcji *sppRx* odczytującej

Listing 3. Funkcja demo dla mikrokontrolerów STM32F103 i SAM7

```
void demo(void)
{
    uint8_t key = debug_inkey();
    switch(key)
    {
        case ‚0’:
            txPacket.dstAddr = 0x40;
            txPacket.len = 1;
            txPacket.payload[0] = SPP_ID_CHAN_TOGGLE | 0x0;
            if(sppTx(&txPacket)) xprintf("sppTx OK\n");
            else xprintf("sppTx failed\n");
            break;
        case ‚1’:
            txPacket.dstAddr = 0x40;
            txPacket.len = 1;
            txPacket.payload[0] = SPP_ID_CHAN_TOGGLE | 0x1;
            if(sppTx(&txPacket)) xprintf("sppTx OK\n");
            else xprintf("sppTx failed\n");
            break;
    }
    if(sppRx(&rxPacket))
    {
        xprintf("rx: src=%x, dst=%x, len=%x, payload=",
        rxPacket.srcAddr,
        rxPacket.dstAddr,
        rxPacket.len);
        for(int i=0;i<rxPacket.len;i++) xprintf("%x ",
        (rxPacket.payload[i]));
        xprintf(„\n”);
    }
}
```

nadchodzące pakiety i ewentualnie także funkcji *sppIdleTimeout*.

Gdy opracowujemy własny port dla protokołu SPPoB i chcielibyśmy trzymać się dotychczasowej konwencji w organizacji plików, to będzie trzeba:

- utworzyć w katalogu głównym nowy plik nagłówkowy *spp_[nazwaplatformy].h* (można także skopiować już istniejący i później go modyfikować),
- utworzyć lub podmienić w tym pliku funkcje specyficzne dla platformy,
- dodać odpowiednie dyrektywy preprocesora w pliku *spp.c* (zob. **listing 1**) tak, aby dało się wykryć naszą platformę np. na podstawie globalnej definicji specyficznej dla używanego mikrokontrolera.

Podsumowanie i dalsze pomysły

Działanie SPPoB zostało zweryfikowane w praktyce w sieci działającej nieprzerwanie od wielu miesięcy i składającej się z ponad 20 urządzeń, w tym klawiatur, sterowników oświetlenia oraz różnego rodzaju czujników. Fizyczną rozpiętość sieci testowej można szacować na ponad 100 m.

SPPoB w wielu zastosowaniach może zastąpić protokół Modbus, mimo, iż SPPoB nie jest aż tak bardzo „niezniszczalny” (tu występuje prawdopodobieństwo kolizji, choć jest niewielkie). Rekompensatą jest tutaj obsługa trybu „multimaster”, dzięki której uzyskuje się świetną decentralizację sterowania urządzeniami w sieci.

Oczywistym mankamentem sieci opartych na RS-485 jest to, że zwarcie w dowolnym miejscu magistrali powoduje unieruchomienie sieci. Tutaj po prostu trzeba na to uważać. Rozwiązaniem zmniejszającym skutki takich zdarzeń mógłby być „repeater”, który będzie przekazywał dane (choćby 1:1) pomiędzy różnymi segmentami sieci, odpowiadającym np. różnym piętrom budynku lub różnym pokojom. Wtedy zwarcie w jednym segmencie nie spowoduje stanu awarii w całej sieci. Przy okazji układ taki mógłby filtrować pakiety, które mogą pozostać „po jednej stronie”, tj. nie muszą przechodzić do innych segmentów. Jednak wtedy jego logika działania stanie się nieco bardziej skomplikowana. Wspomniany „repeater” mógłby rozszerzyć także maksymalną liczbę urządzeń na magistrali wynikającą z fizycznych ograniczeń transceiverów RS-485.

Nie powinno być też problemu z podłączeniem sieci SPPoB do Internetu lub sieci komórkowych. W tym celu potrzebny będzie odpowiedni układ mostu pomiędzy tymi dwoma rodzajami sieci. Do połączenia z Internetem powinien wystarczyć mały serwer na komputerze wbudowanym wraz z przejściówką USB-RS485. Implementacja SPPoB na komputerze nie jest bardzo trudna. Natomiast interfejs do sieci GSM lub GPRS uzyskamy stosując tani modem (np. SIM900) wraz z prostym mikrokontrolerem wyposażonym w dwa interfejsy UART/USART (jeden do komunikacji z modemem, a drugi do interakcji z urządzeniami SPPoB). Pierwotnie właśnie do tego celu przeznaczony był identyfikator pakietu zawierającego ciąg znaków ASCII – można go użyć do przesyłania pakietów zawierających treść wiadomości SMS.

Na razie nie podaję gotowych rozwiązań urządzeń „złóż, zaprogramuj i działaj” pracujących z SPPoB, ponieważ z praktyki wiem, że są to urządzenia, których forma i funkcjonalność jest bardzo ściśle zależna od indywidualnych upodobań projektanta systemu. Jeśli pojawi się zainteresowanie urządzeniami z SPPoB, to postaram się przedstawić kilka w miarę uniwersalnych rozwiązań takich jak włączniki, klawiatury lub uniwersalne moduły, do których można dodać układy wykonawcze lub czujniki.

Robert Brzoza-Woch
robert.brzoza@gmail.com