SPPoB – prosty protokół pakietowy dla automatyki inteligentnego budynku (1)

W artykule chciałbym podzielić się z Czytelnikami pomysłem na prosty protokół komunikacyjny o roboczej nazwie SPPoB z ang. Simple Packet Protocol over Bus. Jego podstawową cechą jest prostota i idąca za nią możliwość łatwego „ogarnięcia" dla elektronika. W założeniach podstawowym zastosowaniem protokołu miała być „mała automatyka" inteligentnego budynku, w której urządzenia połączone są magistralą RS-485 z wieloma układami mogącymi pracować jako nadrzędne. Może on mieć zastosowanie do obsługi różnorodnych urządzeń domowych, transmisji danych z czujników, rozsyłania aktualnego stanu centralki alarmowej lub nawet przesyłania krótkich informacji tekstowych. Sam protokół SPPoB lub jego wybrane elementy (np. struktura pakietu, pomysł na adresację) mogą zostać wykorzystane do wielu innych zastosowań.

System automatyki inteligentnego budynku można relatywnie łatwo zestawić stosując dostępne w handlu moduły transmisji bezprzewodowej. Najtańsze moduły radiowe umożliwiają wysłanie bądź odbiór informacji praktycznie bez ich przetwarzania, tj. „bit po bicie". Wtedy pojawia się problem szyfrowania przesyłanych informacji, które komplikuje program poszczególnych urządzeń oraz kosztuje dodatkową moc obliczeniową i zasoby mikrokontrolera. Współcześnie uchodzi za niedopuszczalne używanie do sterowania urządzeniami nieszyfrowanej komunikacji radiowej. Jest to o tyle uzasadnione, że bez szyfrowania praktycznie nie mamy kontroli ani kto podsłuchuje transmisję, ani kto wprowadza własnym nadajnikiem mylne dane.

Po drugiej stronie mamy rozwiązania przewodowe. W tym przypadku komunikacja – w przeciwieństwie do bezprzewodowej – może zostać „zamknięta" w obrębie jednego mieszkania czy budynku. Podczas gdy nieszyfrowaną transmisję radiową można prosto przechwycić, dostęp osób nieuprawnionych do okablowania budynku jest dużo trudniejszy. Najtańsze rozwiązania przewodowe mają jednak oczywistą wadę: konieczność prowadzenia dodatkowego okablowania sterującego. Mimo, że nie musi ono być drogie, to jednak bywa kłopotliwe w instalacji w już istniejących obiektach. Dlatego warte rozważenia są rozwiązania oparte na komunikacji przez linie energetyczne. Są one o tyle wygodne, że pomiędzy urządzeniami nie trzeba prowadzić dodatkowych przewodów. Z drugiej strony efektywna transmisja danych siecią energetyczną to zagadnienie wymagające zastosowania specjalizowanych układów lub implementacji zaawansowanych algorytmów przetwarzania sygnałów. Jedno i drugie rozwiązanie nie należy do tanich.

Proponowane rozwiązanie

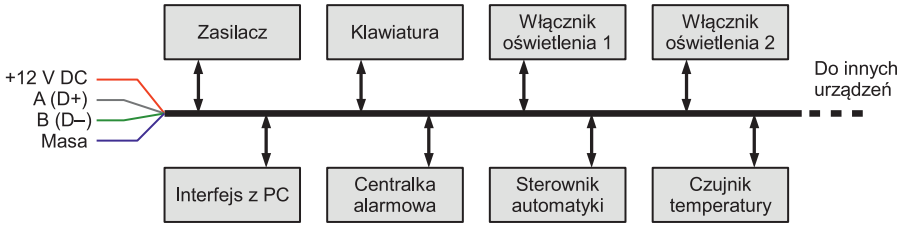
W artykule opisuję protokół SPPoB, którego głównym założeniem jest prostota i idące za nią pozytywne następstwa: brak przywiązania do konkretnego producenta, bardzo małe wymagania sprzętowe, a stąd niska cena. Wadą natomiast jest konieczność prowadzenia przewodu sterującego pomiędzy urządzeniami. Jest to „zło konieczne" potrzebne do utrzymania jak najniższej ceny punktu. Za „punkt" można uznać pojedyncze urządzenie współpracujące z innymi, np. przycisk włączający światło, układ załączający odbiorniki energii elektrycznej, czujnik ruchu itp. Urządzeń tych może być całkiem dużo, więc zmniejszenie ich kosztów jednostkowych może mieć znacznie w skali całego mieszkania bądź budynku.

Warstwa fizyczna i transmisja danych

Protokół SPPoB został opracowany dla magistrali RS-485, przez którą dane transmitowane są asynchronicznie w trybie half-duplex. Proponowane i testowane w praktyce parametry transmisji dla interfejsu RS-485 to 19200 baud, 8 bitów danych, brak bitu parzystości, 1 bit stopu. Są to bardzo typowe i łatwe do uzyskania nastawy układów peryferyjnych USART bądź UART nawet w najprostszych mikrokontrolerach lub na komputerze PC.

Skoro i tak trzeba poprowadzić okablowanie sterujące, warto zastosować przewód 4-żyłowy. Wówczas magistrala RS-485 (sygnały A/D+ i B/D-) zajmie dwa przewody, a pozostałe dwa możemy przeznaczyć na wspólną masę i zasilanie poszczególnych urządzeń. W praktyce bardzo wygodne jest rozprowadzenie napięcia +12 V – wówczas obniżymy je w urządzeniach do +5 V lub +3,3 V, a samo napięcie +12 V przydaje się choćby do sterowania przekaźnikami elektromagnetycznymi (te o napięciu znamionowym cewki 12 V pobierają mniej prądu niż ich wersje na 5 V). Napięciem 12 V można także zasilać popularne elementy systemów alarmowych, takie jak czujniki ruchu.

Na **rysunku 1** przedstawiono topologię systemu małej automatyki domowej składającego się z kilku urządzeń z interfejsem RS-485. Dodatkowo rozpisano oznaczenia przewodów wchodzących w skład magistrali danych i zasilania. Do zapewnienia fizycznego połączenia pomiędzy urządzeniami



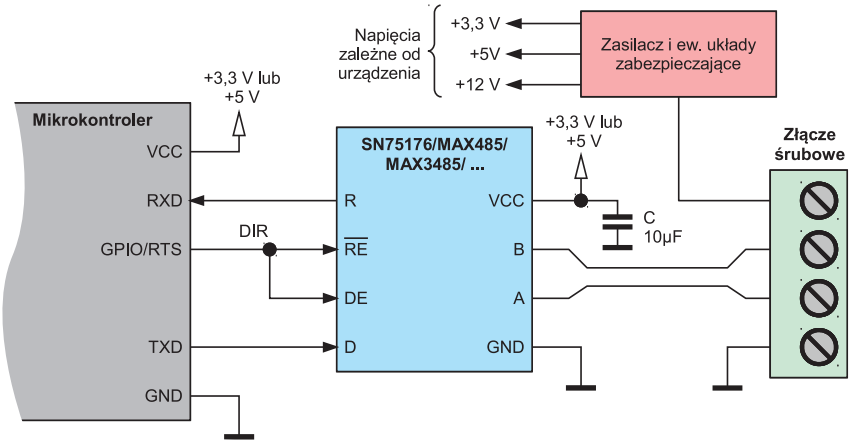
Rysunek 1. Topologia i przykład zestawienia kilku urządzeń komunikujących się protokołem SPPoB przez magistralę RS-485

powinno się zastosować kabel typu skrętka (przynajmniej dla sygnałów magistrali RS-485). W praktyce do transmisji danych i dystrybucji zasilania można rozważyć przewód „domofonowy" YTDY 4x0,5 mm – jest on tani i ogólnodostępny, np. w sklepach z akcesoriami do systemów alarmowych. Przed zestawieniem systemu warto sprawdzić eksperymentalnie, czy wybrany przewód nie sprawi problemów, szczególnie na dużych odległościach.

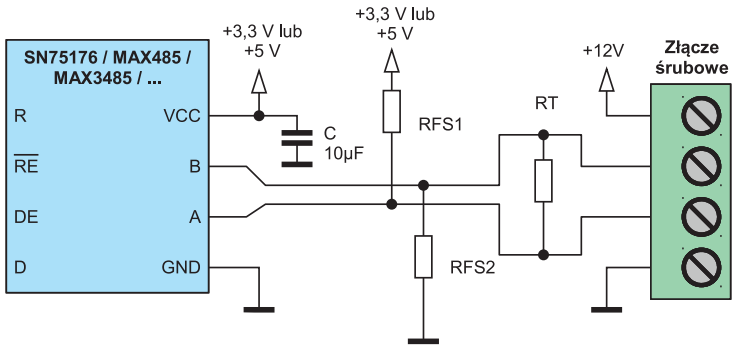
Fizyczne podłączenie mikrokontrolera do magistrali RS-485 nie jest niczym nowym i można je zrealizować na przykład według **rysunku 2**. Zaznaczone sygnały *RXD* i *TXD* to odpowiednio wejście odbiornika i wyjście nadajnika interfejsu UART lub USART (*Universal Synchronous-Asynchronous Receiver-Transmitter*). Dodatkowo potrzebne też jest wyprowadzenie *DIR* sterujące kierunkiem transmisji. Zależnie od mikrokontrolera, wyprowadzenie to trzeba będzie oprogramować jako cyfrowe wyjście ogólnego przeznaczenia (np. w przypadku AVR) lub, alternatywnie, uruchomić je jako dedykowane wyjście interfejsu USART do sterowania kierunkiem transmisji (może być tak skonfigurowane wyjście RTS modułów USART w mikrokontrolerach rodziny AT91SAM). W przedstawionej na rysunku 2 konfiguracji, jako transceiver interfejsu RS-485 używany jest jeden z powszechnie dostępnych układów scalonych kompatybilnych wyprowadzeniami z SN75176B. Choć układ SN75176B jest bardzo tani, to jednak ma on szereg mankamentów, z których najbardziej odczuwalnym jest wysoki pobór prądu. Dlatego można rozważyć zastosowanie jego nieco droższego zamiennika o wyraźnie niższym poborze prądu, np. MAX485, ADM485 (zasilanie +5 V) lub MAX3485 (zasilanie +3,3 V). Transceivery RS-485 mają także różne rezystancje wejściowe, lecz typowo spotykana wartość to 12 kΩ. W takiej konfiguracji (przynajmniej teoretycznie) możemy mieć do 32 odbiorników na magistrali. Można również zastosować np. układ MAX487 o rezystancji wejściowej 48 kΩ (1/4 obciążenia), co podnosi wspomniane ograniczenie do 128 urządzeń.

Zaznaczony na rysunku 2 kondensator odsprzęgający C (np. typu MLCC) o wartości 10 µF powinien być zamontowany jak najbliżej wyprowadzeń układu transceivera i jest wbrew pozorom dość istotnym elementem. Transceivery RS-485 mogą bowiem chwilami pobierać duży prąd i pominięcie tego kondensatora może spowodować nieprzewidywalne zachowanie układu. Jeszcze lepiej byłoby odsprzęgać zasilanie równolegle połączonymi dwoma kondensatorami 10 µF i 100 nF dla dodatkowej filtracji zakłóceń o wyższych częstotliwościach.

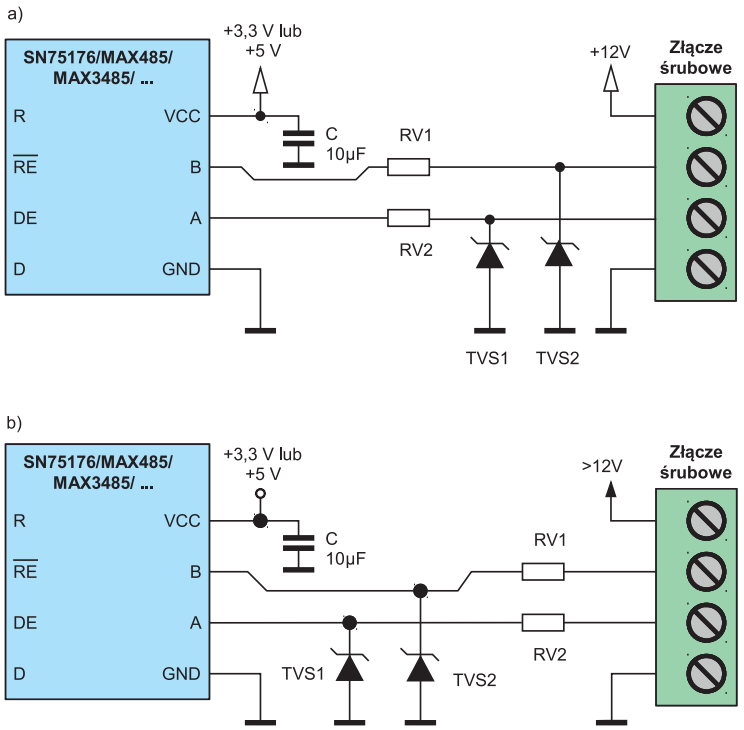
Magistralę RS-485 z układu transceivera wygodnie jest wyprowadzić wprost na złącza śrubowe nazywane „terminal block" (jak na rysunku 2). W przypadku urządzeń znajdujących się na końcach magistrali, można rozważyć zastosowanie terminatorów (*RT* na **rysunku 3**), czyli rezystorów zapobiegających odbiciom sygnału. W teorii RT powinny mieć wartość równą impedancji charakterystycznej zastosowanego przewodu i w trybie half-duplex powinno się je stosować na obu końcach magistrali RS-485. Często przyjmo-



Rysunek 2. Ogólny schemat podłączenia mikrokontrolera do magistrali RS-485



Rysunek 3. Podłączenie rezystorów fail-safe (RFS1, RFS2) oraz terminatora (RT)



Rysunek 4. Układy zabezpieczeń przeciwprzepięciowych dla magistrali RS-485: a) tradycyjny b) ograniczający skutki przypadkowego zwarcia linii A i B do szyny zasilania przy napięciu wyższym niż maksymalne dopuszczalne dla linii A i B transceivera

waną wartością RT jest 120 Ω. W praktyce, przy niskich prędkościach transmisji (jak w SPPoB) i długościach magistral rzędu kilkudziesięciu metrów, stosowanie terminatorów nie jest bezwzględnie konieczne.

Co prawda bez rezystora RT można uzyskać poprawną komunikację, ale już pominięcie rezystorów RFS1 i RFS2 (z rysunku 3) może mieć negatywne skutki. W momencie, gdy żadne urządzenie nie wysyła danych, magistrala RS-485 w konfiguracji half-duplex jest narażona na przypadkowe zakłócenia, które na pewno wcześniej czy później wygenerują fałszywe znaki odbierane przez dołączone mikrokontrolery. Tej sytuacji mają zapobiegać rezystory *fail-safe*, RFS1 i RFS2. Ich wartość należy tak dobrać, by w połączeniu z terminatorem uzyskać na magistrali RS-485 napięcie różnicowe $V_A - V_B >$ 200 mV, co oznacza stan logiczny „1”. Dla RT=120 Ω i napięcia zasilającego 5 V, RFS1 i RFS2 powinny mieć wartość ok. 1440 Ω. Nowoczesne transceivery mają wbudowane zabezpieczenie przed stanami nieustalonymi w postaci przesuniętych progów detekcji napięcia wejściowego w odbiorniku. Jednakże projektując własne urządzenie z interfejsem RS-485 warto uwzględnić na płytce możliwość zamontowania rezystorów *fail-safe* w razie, gdybyśmy mieli zamiar wlutować klasyczny układ transceivera.

Dodatkowe zabezpieczenia

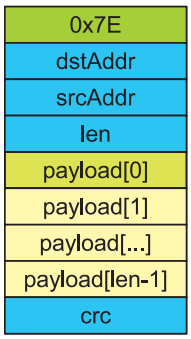
Jeśli chcemy, by nasza sieć domowa stała się bardziej odporna na przepięcia i przypadkowe zwarcia, możemy rozważyć zastosowanie dodatkowych zabezpieczeń przeciwprzepięciowych na magistrali RS-485 oraz w lokalnych zasilaczach poszczególnych urządzeń. Dalej omówiono dwa rodzaje zabezpieczenia magistrali. Na **rysunku 4a** widzimy typowy sposób zabezpieczenia magistrali RS-485 przed przepięciami. W razie wystąpienia chwilowego impulsu wyższego napięcia na jednej z linii, specjalne diody tłumiące przepięcia (TVS od *transient-voltage-suppression* lub *Transil*) blokują wystąpienie napięć wyższych niż maksymalne dopuszczalne, przez co „biorą na siebie” całą nadmiarową energię, nawet jeśli jest to tylko krótki impuls. Dodatkowo, rezystory RV1 i RV2 mogą wtedy odciążyć wewnętrzne układy zabezpieczeń wbudowane w transceiver. Mniej typowym sposobem zabezpieczenia jest zastosowanie rezystorów RV przed diodami TVS, tj. od strony interfejsu RS-485, jak na **rysunku 4b**. Taki sposób zabezpieczenia może być przydatny, gdy zastosujemy transceivery o maksymalnym akceptowanym napięciu na wyprowadzeniach A i B wynoszącym mniej niż planujemy zastosować do zasilania urządzeń sieci. W razie zwarcia sygnału A lub B z napięciem zasilania w pesymistycznym scenariuszu moglibyśmy stracić nawet wszystkie transceivery na magistrali. W przypadku zasilania z +12 V raczej nie powinno być tego problemu, ponieważ większość transceiverów RS-485 według producentów radzi sobie z takimi napięciami. Przy wyższych napięciach (lub żeby po prostu spać spokojniej) można zastosować układ z rysunku 4b, w którym rezystory RV1 i RV2 pełnią rolę chwilowych „promienników” energii, a w ostateczności bezpieczników. Jeśli zastosujemy rezystory o wartości rzędu kilkudziesięciu do kilkuset omów w rozmiarze 0805 (typowo mają moc znamionową ok. 0,125 W), to nie wpłyną one znacząco na transmisję danych, a zyskamy możliwość uratowania transceiverów w razie przypadkowego, krótkiego zwarcia. Po dłuższym czasie rezystory RV najprawdopodobniej zadziałają jak bezpieczniki i po prostu przepalą się. Jeśli byśmy nie zamontowali RV (zwarcie w ich miejscach), wtedy albo zadziała zabezpieczenie przeciwzwarciowe w zasilaczu, albo nadmiar energii wydzieli się na diodach TVS i spotka je ten sam los, tylko po nieco dłuższej chwili zwarcia. Należy pamiętać, że przepięcie może pojawić się także na linii zasilającej. Tam również można zrobić użytek z diod TVS wraz z szeregowo dołączonymi bezpiecznikami polimerowymi PTC.

W praktyce nie ma potrzeby stosowania „kuloodpornych” zabezpieczeń w każdym urządzeniu na magistrali. Gdy mamy kilka urządzeń umieszczonych w niedużej odległości od siebie, powinno wystarczyć zamontowanie zabezpieczeń tylko w jednym z nich (w końcu i tak wszystkie będą połączone wspólnymi czterema przewodami). Dodatkowo, duża ilość „byle jakich” diod TVS będzie także wnosiła pasożytnicze pojemności. Zależnie od ich napięcia znamionowego będą to pojemności rzędu pojedynczych nanofaradów na diodę.

Mając zestawioną sieć na magistrali RS-485 i wiedząc, jak odpowiednio podłączyć do niej mikrokontrolery, możemy przejść do sedna sprawy, czyli danych, które chcemy nią przesyłać.

Struktura pakietu

Zgodnie z zapowiedziami, proponowana struktura pakietu dla protokołu SPPoB jest bardzo prosta. Pakiet ma typową strukturę i korzysta z kilku rozwiązań powszechnie wykorzystywanych przy przesyłaniu danych. Ogólną strukturę pakietu przedstawiono na **rysunku 5**.



Rysunek 5. Struktura pakietu w protokole SPPoB

Początek pakietu to bajt *0x7E* (posługujemy się zapisem szesnastkowym jak w języku C/C++). Pojawienie się bajtu *0x7E* na magistrali ma zresetować maszyny stanów we wszystkich odbiornikach i przełączyć je w tryb rozpoczęcia odbioru pakietu. Zaraz za bajtem początku jest bajt *dstAddr* (od *destination address*) zawierający adres docelowy urządzenia, które ma odebrać pakiet. Nadawca pakietu wpisuje swój adres do pola *srcAddr* (od *source address* czyli „adres źródłowy”). Zakładamy, że każde urządzenie na magistrali ma swój niepowtarzalny adres. Ilość bajtów właściwych danych określa kolejne pole, *len* (od *length*), a treść pakietu umieszczana jest w bajtach oznaczonych jako *payload[0]*, *payload[1]*, itd. Poprawność danych można sprawdzić na podstawie 8-bitowej sumy kontrolnej umieszczanej w polu ***crc***. Suma kontrolna to CRC-8 liczone w taki sam sposób jak dla popularnego protokołu 1-Wire. To jest oczywiście tylko punkt wyjścia i we własnej implementacji wielomian sumy kontrolnej można dowolnie zmienić, dostosowując go do indywidualnych potrzeb.

U większości Czytelników pewnie już dawno pojawiła się wątpliwość, co zrobić, jeśli chcemy przesłać bajt o wartości *0x7E*? W końcu przesłanie go „ot tak”, według wcześniejszych informacji, spowoduje zresetowanie wszystkich odbiorników na magistrali. To prawda – *0x7E* jest bajtem zarezerwowanym tylko dla początku pakietu. Jeśli na dowolnym innym miejscu pakietu ma się znaleźć ta liczba, należy ją „rozbić” na ciąg: *0x7D* i dalej *0x5E*. A co z *0x7D*? Podobnie – rozbijamy go na *0x7D* i *0x5D*. Podsumowując:

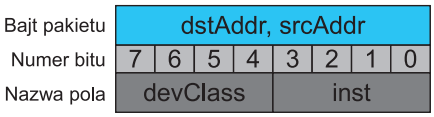
- chcemy przesłać *0x7E* – przesyłamy *0x7D* i *0x5E*,
- chcemy przesłać *0x7D* – przesyłamy *0x7D* i *0x5D*.

Oczywiście odbiornik ma działać symetrycznie, czyli rozpoznawać sekwencje *0x7D-0x5E* oraz *0x7D-0x5D*, a następnie zamieniać je na, odpowiednio, *0x7E* i *0x7D*. Wspomniane rozbijanie i scalanie liczb jest oczywiście niewidoczne dla programu głównego – nie interesuje nas sposób transmisji, tylko gotowe dane.

Skąd akurat takie liczby? Pomysł na przesyłanie sekwencji „escape” nie jest nowy i korzysta z niego wiele poważnych protokołów. Akurat liczby *0x7E* dla początku ramki (*frame delimiter*) oraz *0x7D* dla znaku modyfikującego „escape” stosuje się m.in. w protokole High-Level Data Link Control, czy w systemie TinyOS dla radiowych sieci sensorowych.

Listing 1. Struktura danych odwzorowująca pakiet protokołu SPP

```
typedef struct __attribute__((packed))
{
    uint8_t dstAddr;
    uint8_t srcAddr;
    uint8_t len;
    uint8_t payload[SPP_PAYLOAD_LEN];
}T_sppPacket;
```



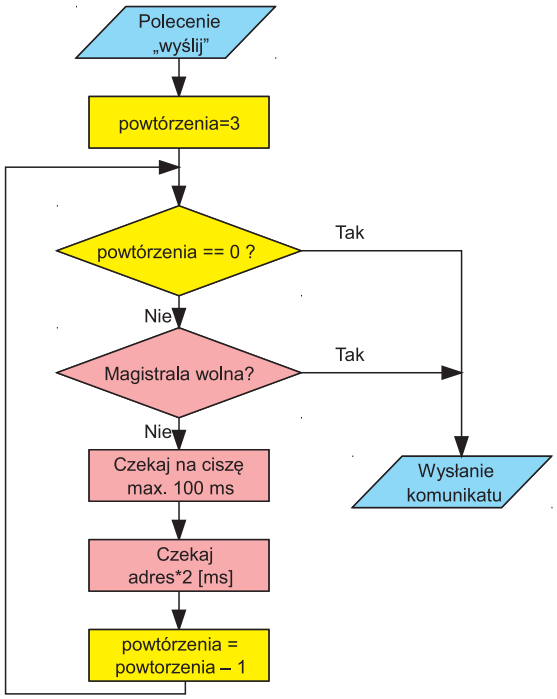
Rysunek 6. Pola bitowe w bajtach adresu

Odwzorowanie omówionego wyżej pakietu na typ w języku C zostało przedstawione na **listingu 1**. Aby utworzyć „obiekt" reprezentujący pakiet, wystarczy skorzystać z typu *T_sppPacket* i dalej możemy bezpośrednio wpisywać dane do jego odpowiednich pól. W opracowanych, przykładowych implementacjach wysyłanie pakietu powoduje automatyczne uzupełnienie pola *srcAddr* adresem na stałe przypisanym dla danego urządzenia. Dla programisty „niewidoczne" są pola bajtu startowego i sumy kontrolnej CRC, ponieważ niepoprawne CRC spowoduje w normalnym działaniu po prostu odrzucenie pakietu przy odbiorze, a bajt startu jest zawsze taki sam. Dodatkowy atrybut („packed") przy definicji struktury danych to informacja dla kompilatora, aby niezależnie od architektury mikroprocesora układał poszczególne pola struktury w pamięci jeden za drugim. Bez tego atrybutu kompilatory GCC dla mikrokontrolerów ARM prawdopodobnie wyrównałyby każdy bajt do 32-bitów, co może nie być pożądane przy operacjach kopiowania fragmentów pamięci.

Adresowanie urządzeń

Wspomniałem już, że każde urządzenie komunikujące się protokołem SPPoB ma swój unikalny adres. Z racji dość ograniczonej liczby transceiverów na magistrali RS-485 (do 128) raczej nie zajdzie potrzeba adresowania większej liczby urządzeń i stąd adresy mają rozmiar jednego bajtu. Dodatkowo adresy są podzielone na 4-bitowe pola *devClass* (bity 7:4) i ***inst*** (bity 3:0) jak na **rysunku 6**. Pole *devClass* (od *device class*)oznacza numer „klasy" urządzenia. Klasą urządzenia może być: klawiatura włączająca oświetlenie, czujnik temperatury, sterownik włączający oświetlenie, sterownik włączający ogrzewanie itd. Natomiast ***inst*** (od *instance*, wystąpienie) to numer urządzenia w danej klasie. I tak sterowniki oświetlenia jednego rodzaju mogą mieć np. numer klasy *devClass=0x4* i może ich być 15 (od *inst=0x0* do *inst=0xE*). Wtedy będziemy wiedzieć, że urządzenia o adresach od *0x40* do *0x4E* są włącznikami światła, a np. od *0x20* do *0x2E* to klawiatury itd.

Nasuwa się pytanie – po co ten podział? Czyż nie można przypisać każdemu urządzeniu kolejnego adresu, nieważne czym jest to urządzenie? I wreszcie – dlaczego liczymy *inst* od *0x0* do *0xE*? Wszystkie powyższe „komplikacje" mają na celu łatwiejsze rozsyłanie danych rozgłoszeniowych. W SPPoB przewidziane są dwa rodzaje rozgłoszeń: do wszystkich urządzeń w sieci (*broadcast*) oraz do wszystkich urządzeń danej klasy (prosty mechanizm *multicast*). Możemy wysłać pakiet z polem *dstAddr = 0xFF* i wtedy odbierze go każde urządzenie na magistrali. Jeśli natomiast wyślemy pakiet z polem *inst=0xF* w *dstAddr*, to otrzyma go każde urządzenie wybranej klasy. W powyższym przykładzie: *dstAddr=0x4F* będzie oznaczał rozesłanie informacji do wszystkich sterowników oświetlenia.



Rysunek 7. Algorytm wysyłania z minimalizacją prawdopodobieństwa kolizji

Kolizje na magistrali

Z racji, że magistrala RS-485 jest dzielona przez wszystkie urządzenia komunikujące się przez protokół SPPoB, należało wprowadzić jakiś sposób radzenia sobie z sytuacją, gdy więcej niż jedno urządzenie będzie chciało wysłać dane. Mechanizm działania jest dość prosty. Każde urządzenie bez przerwy obserwuje transmisję na magistrali i każdy odebrany bajt musi być sprawdzony (choćby po to, żeby wykryć bajt początku ramki, *0x7E*). Początek odbioru jakiegokolwiek pakietu jest tożsamy z zajęciem magistrali. Z tego powodu każde urządzenie śledzi odbiór całego pakietu, niezależnie czy jest jego odbiorcą. Jeśli urządzenie jest odbiorcą pakietu, zbiera informacje z magistrali do tymczasowego bufora. W przeciwnym przypadku urządzenie tylko sprawdza długość pakietu z pola *len* i „odlicza sobie", ile jeszcze bajtów pozostało do końca pakietu. Zajętość magistrali rozpoczyna się od odebrania bajtu *0x7E*, dalej adresów, długości, określonej liczby bajtów *payload* i wreszcie odebrania sumy kontrolnej.

Teoretycznie jest jeden newralgiczny moment narażenia na kolizję, gdy jedno urządzenie na magistrali transmituje już znak startu *0x7E*, a drugie jeszcze nie zdąży go odebrać i także rozpocznie nadawanie „myśląc", że magistrala była wolna. To niebezpieczne okno czasowe trwa mniej więcej tyle, ile wynosi czas wysyłania jednego znaku przez port szeregowy, czyli w przypadku SPPoB ok. 521 µs (zakładamy, że znak składa się z 10 bitów, a szybkość transmisji to 19200 baudów, stąd 10 bitów / 19200 bitów/s ≈ 521 µs).

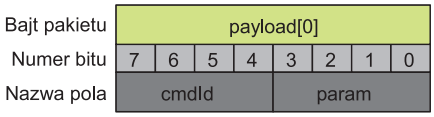
Mimo, że prawdopodobieństwo kolizji przy takich parametrach jest bardzo małe, to jednak w urządzeniach krytycznych warto rozważyć mechanizm potwierdzeń lub kontrolnego „odpytania" sterowanego urządzenia, czy dany parametr na pewno został poprawnie ustawiony.

Wywołanie funkcji wysyłającej pakiet spowoduje jego natychmiastowe wysłanie tylko wtedy, gdy magistrala jest wolna. Natomiast, gdy będzie ona w tym momencie zajęta, rozpocznie się sekwencja:

1. oczekiwania na zwolnienie magistrali,
2. po wykryciu ciszy odczekanie czasu zależnego od adresu urządzenia (zabezpieczenie przed próbą jednoczesnej transmisji z kilku oczekujących urządzeń na raz),
3. sprawdzenie zajętości magistrali bez czekania,
4. jeśli wolna – wysłanie pakietu, w przeciwnym wypadku powrót do punktu 1.

Dodatkowo wprowadzono ograniczenie ilości powtórzeń powyższego algorytmu do trzech. Po trzech nieudanych próbach uzyskania wolnej magistrali możemy założyć, że jej zajętość najprawdopodobniej jest wynikiem błędnego (np. niedokończonego) pakietu i rozpocząć nadawanie tak czy inaczej. Algorytm ten przedstawiono na **rysunku 7**.

Istnieje także mechanizm programowego *time-outu* dla stanu zajętości magistrali. Nawet, gdy urządzenie nie będzie próbowało wysyłać danych, a magistrala będzie zajęta przez czas dłuższy niż zdefiniowany w projekcie, nastąpi wymuszenie wyzerowania maszyny stanów modułu odbiornika. Dzięki



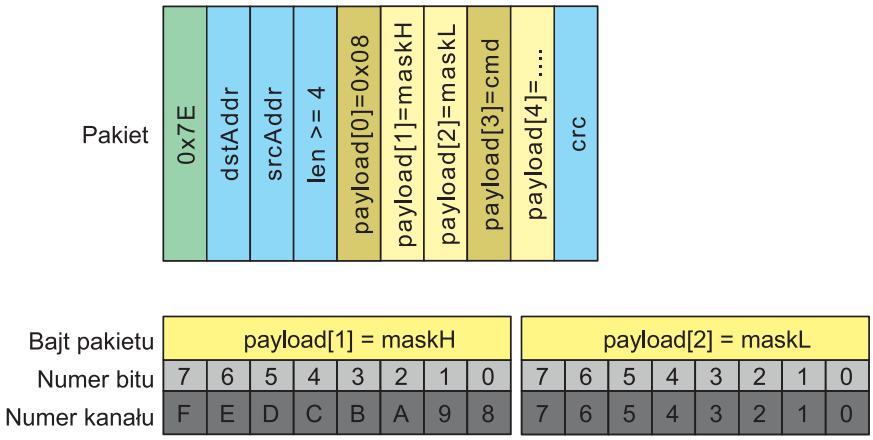
Rysunek 8. Pola bitowe w payload[0]

temu stan „fałszywej zajętości" (wynikający np. ze wspomnianej niedokończonej transmisji pakietu) może zostać wyeliminowany jeszcze przed zgłoszeniem chęci wysyłania pakietu i samo wysyłanie odbędzie się bez zbędnego opóźnienia wynikającego z przejścia przez cały algorytm z rysunku 7.

Na tym etapie mamy zdefiniowany protokół komunikacji pakietowej SPPoB wraz z mechanizmem minimalizacji prawdopodobieństwa kolizji. W dalszej części artykułu zajmiemy się wyższymi warstwami protokołu, czyli tym, co znajdziemy w zawartości pakietu. Innymi słowy zostawiamy „pod spodem" sposób wysyłania i odbierania pakietu i koncentrujemy się na tym, co można nim przesłać.

Bajt identyfikacji i komendy

Pakiet, który dotarł do pewnego urządzenia, powinien nieść informację, co zawiera. Prosty przykład. Mamy włącznik światła, w którym możemy przełączyć stan jednego z kanałów wyjściowych (taśma LED nr 1, taśma LED nr 2), wyłączyć wszystkie kanały, ustawić jasność poszczególnych kanałów itd. Ale przydatne byłoby także, gdybyśmy mogli zapytać urządzenie „jaka jest jasność w kanale nr 1?". Dlatego warto wyróżnić pierwszy bajt pola *payload* (czyli *payload[0]*) tak, aby mówił, co zawiera pakiet: czy jest to żądanie włączenia wybranego kanału, jego przełączenia, ustawienia jego wartości, czy może zapytanie o uprzednio ustawioną jasność?



Rysunek 9. Pakiet MULTICHANNEL. Poszczególne bity w maskH i maskL odpowiadają kanałom od 0x0 do 0xF, payload[3] zawiera komendę cmd do wykonania (cmd może zająć więcej niż 1 bajt)

W **tabeli 1** zestawiono przykładowe wartości i odpowiadające im czytelne nazwy komend i identyfikatorów pakietu. W przykładowej implementacji zawarto plik nagłówkowy zawierający te nazwy w formie makrodefinicji. W głównej pętli własnego programu można po prostu porównać wartość *payload[0]* z odpowiednią makrodefinicją.

Bajt *payload[0]* możemy sobie logicznie podzielić na dwa pola bitowe, jak na **rysunku 8**. Bity 7:4 możemy traktować jako identyfikator komendy (***cmdId***), natomiast bity 3:0 to parametr komendy (***param***). Np. polecenie „włącz kanał nr 3" będzie się składało z komendy *SPP_ID_CHAN_ON* (liczba *0x10*)

REKLAMA

Jesteś mobilny? My również.

Wydanie papierowe
Portal automatykaB2B.pl
Cyfrowe e-wydanie
Wydanie dla iPada
Strona mobilna

Miesięcznik APA dostępny jest jako wydanie papierowe oraz w kilku wersjach cyfrowych.

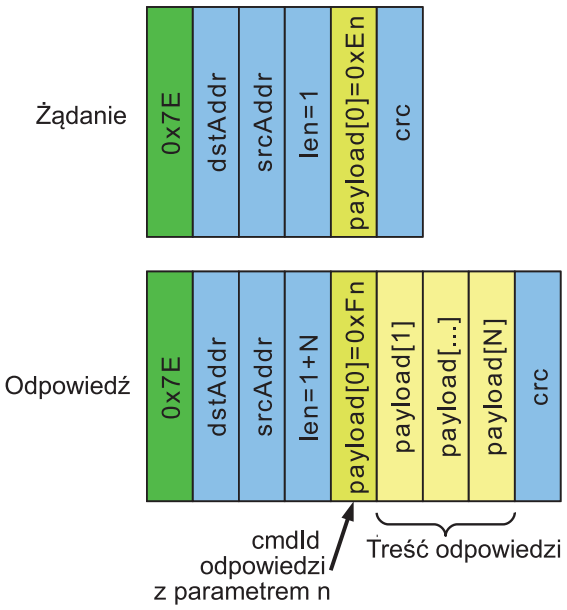
Tabela 1. Wybrane nazwy komend i identyfikatory pakietów SPPoB

Kategoria komendy / identyfikatora	Nazwa komendy / identyfikatora w payload[0]	Kod	Ustawienie len	Opis
Globalne i wielokanałowe	SPP_ID_ACK	0x00	1	Pakiet jest potwierdzeniem
	SPP_ID_ALL_ON	0x04	1	Włącz wszystkie kanały w urządzeniu
	SPP_ID_ALL_OFF	0x05	1	Wyłącz wszystkie kanały w urządzeniu
	SPP_ID_ALL_ON_ALT	0x06	1	Włącz wszystkie kanały w urządzeniu (sposób alternatywny, np. powoli rozjaśniając)
	SPP_ID_ALL_OFF_ALT	0x07	1	Wyłącz wszystkie kanały w urządzeniu (sposób alternatywny, np. powoli ściemniając)
	SPP_ID_MULTICHANNEL	0x08	≥4	Komenda w komórkach od payload[3] skierowana do wybranych kanałów, zadanych maskami w payload[1], payload[2]
	SPP_ID_TEXT	0x0A	>2	Komunikat tekstowy; payload[1] zawiera nr identyfikacyjny komunikatu, payload[2,3,...] zawiera tekst kodowany ASCII
Sterowanie kanałów wyjściowych (pojedynczo)	SPP_ID_CHAN_ON	0x1n	1	Włącz kanał „n" (n=0x0...0xF)
	SPP_ID_CHAN_OFF	0x2n	1	Wyłącz kanał „n"
	SPP_ID_CHAN_TOGGLE	0x3n	1	Zmień stan kanału „n"
	SPP_ID_CHAN_ON_ALT	0x4n	1	Włącz kanał „n" w sposób alternatywny
	SPP_ID_CHAN_OFF_ALT	0x5n	1	Wyłącz kanał „n" w sposób alternatywny
	SPP_ID_CHAN_TOGGLE_ALT	0x6n	1	Wyłącz kanał „n" w sposób alternatywny
	SPP_ID_CHAN_UP	0x7n	1	Zwiększ ustawienie kanału (np. rozjaśnij światło)
	SPP_ID_CHAN_DN	0x8n	1	Zmniejsz ustawienie kanału (np. ściemnij światło)
	SPP_ID_CHAN_VALUE	0x9n	2	Zadaj wartość dla wybranego kanału „n" (np. jasność świecenia); wartość wpisujemy do payload[1]
	SPP_ID_CHAN_VALUE_ALT	0xAn	2	J.w., tylko przejście do tej wartości będzie w trybie alternatywnym (np. powoli)
Dane	SPP_ID_TIME	0xD0		Pakiet zawiera aktualny czas (np. do ustawiania i synchronizacji zegarów RTC)
	SPP_ID_CHANNEL_STATE	0xD1		Pakiet zawiera informacje na temat ustawień kanałów urządzenia
	SPP_ID_ALARM_STATE	0xDA		Pakiet zawiera status sensorów centralki alarmowej, przydatny np. do automatycznego włączania światła przez czujniki ruchu
Komunikacja żądanie/odpowiedź	SPP_ID_REQ	0xEn		Żądanie wysłania zestawu informacji „n"
	SPP_ID_RESP	0xFn		Odpowiedź z zestawem informacji „n"

połączonej funkcją OR z parametrem maski kanału *0x03*, co da nam w *payload[0]* liczbę *0x13*. Z racji, że *param* ma 4 bity, wg przedstawionej konwencji możemy zaadresować do 16 kanałów (*0x0* do *0xF*). Jest to liczba wystarczająca dla większości domowych sterowników.

Przydatne bywa sterowanie tylko kilkoma wybranymi kanałami w danym urządzeniu bez modyfikacji stanu innych kanałów. Do tego celu służy komenda *SPP_ID_MULTICHANNEL* (**rysunek 9**). Wpisując ją do *payload[0]* należy zapewnić w *payload[1]* i *payload[2]* maskę bitową mówiącą, do których kanałów chcemy się odnieść („1" na danym bicie oznacza „tak, ten kanał"). Wtedy w *payload[3]* i ewentualnie dalej wpiszemy właściwą komendę dla tych kanałów, np. *SPP_ID_CHAN_ON*, *SPP_ID_CHAN_OFF* itp.

Przewidziano także możliwość komunikacji żądanie-odpowiedź (*request-response*) według **rysunku 10**. Żądanie to pakiet, w którym w bajcie *payload[0]*, w polu *cmdId* wpiszemy *SPP_ID_REQ* (*0xE*), natomiast w polu parametru możemy zdefiniować, o jaki rodzaj danych nam chodzi. Na przykład, czy jest to moc oświetlenia, czas pracy, aktualna godzina, czy zmierzona wartość temperatury. Te informacje kodujemy w pakiecie według uznania w każdym urządzeniu. Odpowiedź ma zostać odesłana w pakiecie o *cmdId* wynoszącym *SPP_ID_RESP* (*0xF*) i parametrze ustawionym na taką samą wartość, jakiej dotyczyło pytanie (to ułatwi ewentualną identyfikację i kolejkowanie odpowiedzi).



Rysunek 10. Pakiety komunikacji żądanie-odpowiedź, n oznacza parametr z zakresu 0x0 do 0xF

Co dalej?

W drugiej części artykułu opiszę przykładowe oprogramowanie, w którym zaimplementowałem prosty sterownik programowy do obsługi SPPoB. Sterownik został tak napisany, by dało się go w miarę łatwo przenieść na inne mikrokontrolery. Programy „demo" są napisane dla trzech modeli mikrokontrolerów: ATmega8 i kompatybilnych, STM32F103 oraz AT91SAM7.

Robert Brzoza-Woch
robert.brzoza@gmail.com