STM32
Zastosowanie interfejsu CAN

Interfejs CAN dzięki odporności na błędy transmisji i dobrą standaryzację znalazł zastosowanie w dziedzinach niezwiązanych bezpośrednio z pojazdami. Współcześnie często są w niego wyposażane urządzenia automatyki. W artykule przybliżymy pokrótce zasadę działania interfejsu CAN, a następnie pokażemy, w jaki sposób nawiązać komunikację, wykorzystując mikrokontrolery z rodziny STM32.

Dodatkowe informacje:
W kontroler magistrali CAN wyposażone są układy STM32 należące do segmentów:
– Performance line – STM32F103
– Connectivity line – STM32F105 i STM32F107

Magistrala CAN może być użyta wszędzie tam, gdzie priorytetem jest bezbłędna komunikacja, natomiast prędkość transmisji nie jest parametrem krytycznym. Standard CAN został stworzony w firmie Bosch w drugiej połowie lat 80., natomiast specyfikacja CAN 2.0 ujrzała światło dzienne w roku 1991. Od tej pory obserwujemy nieprzerwaną ekspansję CAN na nowe obszary zastosowań.

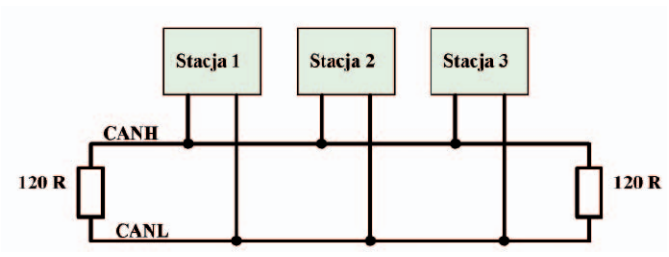
Przyczyny implementacji tego nowego interfejsu w pojazdach miały oczywiście silne korzenie ekonomiczne. Ciągle rosnąca liczba mikrokontrolerów w samochodach powodowała lawinowy wzrost przewodów potrzebnych do nawiązania komunikacji. Konieczne stało się opracowanie sposobu komunikacji opartej o topologię magistrali, która to, przy zwiększeniu szybkości transmisji, pozwoliłaby na zmniejszenie ilości niezbędnych przewodów.

Lata świetności CAN jeszcze nie minęły i jeszcze długo nie miną. Dopóki nie zostanie opracowany standard równie niezawodny, tani i relatywnie prosty w implementacji, dopóty pozycja magistrali CAN będzie niezagrożona. O ile wygląd pojazdów zmienia się nieustannie, o tyle żaden koncern nie będzie wprowadzał zmian w sposobie komunikacji urządzeń, jeśli tylko nie będzie to niezbędne. Pewne nowości można zaobserwować jedynie w pojazdach należących do najwyższych klas.

Magistrala CAN ma wiele zalet, ale też nie jest pozbawiona wad. Jej piętą achillesową jest niewielka przepływność wynosząca maksymalnie 1 Mbit/s. Dyskwalifikuje ją to w niektórych obszarach zastosowań. Z tego powodu prowadzone są badania nad nowymi standardami (np. *FlexRay*).

Magistrala CAN

Magistrala CAN składa się z dwóch przewodów – CANH (zazwyczaj czerwony) i CANL. Informacja kodowana jest sygnałem różnicowym. Gdy pomiędzy dwoma urządzeniami nie występuje zbyt duża różnica potencjałów mas (wg normy mniejsza od 7 V), można wtedy pominąć przewód wspólny, dzięki czemu interfejs fizyczny staje się faktycznie dwuprzewodowy. Wygląd magistrali oraz sposób podłączania kolejnych urządzeń (węzłów) pokazano na **rys. 1**. Na końcach magistrali zawsze są rezystory terminujące, których zadaniem jest ograniczenie zjawiska odbicia sygnału. Najczęściej w roli terminatorów stosuje się rezystory o wartości 120 Ω. Złączem magistrali CAN standardowo może być złącze typu DB9. Rozkład wyprowadzeń jest również objęty standardem.



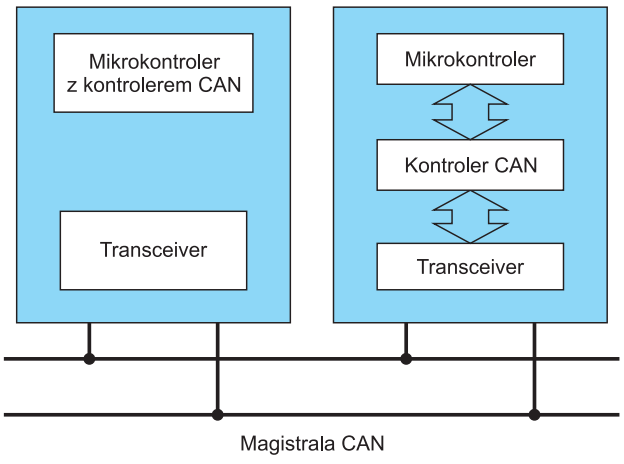
Rys. 1. Topologia magistrali CAN

Typowy węzeł magistrali CAN może składać się z dwóch lub trzech układów scalonych. Jeśli mikrokontroler ma wbudowany kontroler CAN, to wymagane jest jedynie użycie scalonego transceivera (nadajnika/odbiornika). Jego rola polega m.in. na konwersji sygnału różnicowego magistrali CAN do postaci stanów logicznych akceptowalnych przez linie wejścia/wyjścia mikrokontrolera sterującego. Schemat blokowy takich stacji (węzłów) przedstawiono na **rys. 2**. Większość mikrokontrolerów STM32 jest wyposażona w sterownik magistrali CAN, więc do nawiązania komunikacji za pomocą tego medium jest wymagany jedynie wyżej wspomniany transceiver. Ponadto, niektórzy przedstawiciele rodziny STM32 mają wbudowane dwa (zależne) kontrolery CAN.

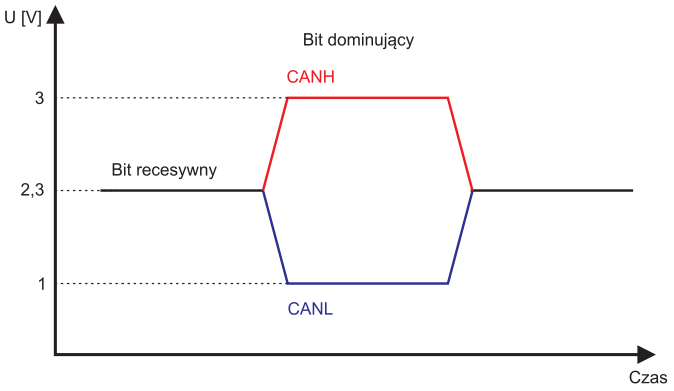
Wartość i kształt sygnałów elektrycznych magistrali zostały zdefiniowane dla dwóch wariantów prędkości przesyłu informacji: szybkiego *high-speed CAN* (do 1 Mbit/s) oraz wolnego *low-speed CAN* (do 125 kbit/s). W artykule będziemy się koncentrować na wersji szybkiej interfejsu.

Początkowo standard definiował ramkę danych zawierającą 11-bitowy identyfikator – jest to wersja standardu CAN 2.0A. Szybko się jednak okazało, że liczba dostępnych identyfikatorów jest zbyt mała. Z tego powodu opracowano aktualizację standardu do wersji CAN 2.0B, którego ramka ma już 29-bitowe pole identyfikatora. Obecnie większość dostępnych kontrolerów magistrali CAN obsługuje obydwa standardy.

Ważnym założeniem podczas opracowywania standardu CAN był wymóg decentralizacji. Magistrala CAN nie ma urządzenia nadrzędnego, jest więc siecią typu *multimaster* – wszystkie elementy mają takie same uprawnienia. Z tego wynika druga charakterystyczna cecha CAN, a mianowicie jest to sieć rozgłoszeniowa (*broadcast networks*).



Rys. 2. Schemat blokowy dwóch rodzajów stacji sieci CAN



Rys. 3. Postać stanów logicznych magistrali CAN dla transceivera SN65HVD230

Fundamentalnym wymogiem spełniającym powyższe założenia jest niedopuszczenie, aby w tym samym czasie wiadomości były nadawane przez więcej niż jedno urządzenie. Tylko jeden węzeł może w danej chwili wysyłać wiadomość, a wszystkie pozostałe muszą być ustawione w tryb nasłuchu.

Wyżej nakreślony problem rozwiązano dzięki odpowiedniej definicji logicznego „0" i logicznej „1". Otóż logiczna „1", określana jako stan recesywny, to różnica napięć na liniach magistrali CANH i CANL wynosząca w idealnym przypadku 0 V. Logiczne „0" to stan dominujący, któremu odpowiada różnica napięć co najmniej 0,9 V (**rys. 3**).

Dzięki takiej definicji stanów logicznych urządzenia podłączone do magistrali mogą rozpoznać, czy w tej samej chwili nadaje jeszcze jakieś inne urządzenie i jeśli zaistnieje taka potrzeba, przerwać nadawanie swojej wiadomości oraz przełączyć się na nasłuch. Od strony logicznej rozwiązanie problemu arbitrażu przyniosło wykorzystanie protokołu wielodostępu CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*). Metoda wielodostępu CSMA/CA wymaga, aby wszystkie urządzenia podłączone do magistrali pracowały z taką samą prędkością.

Ramki

Standard CAN definiuje cztery rodzaje ramek:

- danych,
- przepełnienia,
- żądania transmisji,
- sygnalizacji błędu transmisji.

Podstawowa komunikacja może się odbywać na przykład na zasadzie cyklicznego wysyłania ramek danych. Wystarczy, że urządzenia w określonych odstępach czasu będą wysyłać ramki danych, natomiast w identyfikatorze będzie zakodowany adres odbiorcy informacji.

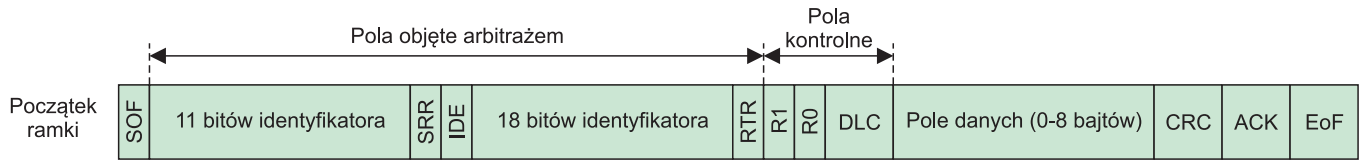
Budowę ramki danych standardu CAN 2.0B pokazano na **rys. 4**. Jak wynika z rysunku, w jednej ramce można przesłać maksymalnie osiem bajtów danych. Początek ramki danych jest określony przez pole SOF (*Start of Frame*), które zawsze jest bitem dominującym. Dalej przesyłane jest pole arbitrażu, a w nim 29-bitowy identyfikator. Za polem sterującym jest miejsce na maksymalnie osiem bajtów danych przeznaczonych do wysłania. Następne pola są związane z procesem wykrywania błędów, który zazwyczaj jest przeprowadzany w sposób sprzętowy przez kontrolery CAN. Na etapie wstępnego poznawania magistrali CAN nie ma potrzeby zagłębiania się w ich rolę.

Niekiedy urządzenie podłączone do sieci nie może czekać na informację. W takiej sytuacji należy źródło żądanych danych odpytać za pomocą ramki żądania transmisji. Jest ona podobna w budowie do ramki danych, jednak pozbawiona pola danych, a bit RTR (*Remote Transmission Request*) jest recesywny, czyli ma wartość logicznej „1". Uproszczoną budowę ramki żądania transmisji pokazano na **rys. 5**.

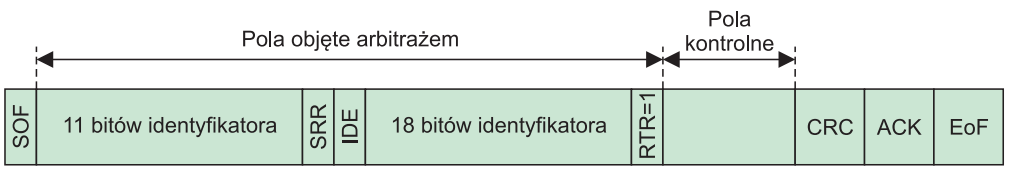
Magistrala CAN jest asynchroniczna, więc do poprawnej pracy wymaga ciągłego zsynchronizowania urządzeń prowadzących wymianę danych. Ma to duże znaczenie w sytuacjach, w których wiele następujących po sobie bitów będzie miało takie same wartości. Bez stosowania dodatkowej metody synchronizacji komunikacja asynchroniczna byłaby w takich warunkach znacznie utrudniona.

Kontrolery CAN po każdym ciągu pięciu takich samych bitów wprowadzają do ramki bit przeciwny (komplementarny), mechanizm taki nazywa się *bit stuffing*. Po odebraniu ramki te dodatkowe bity są automatycznie usuwane.

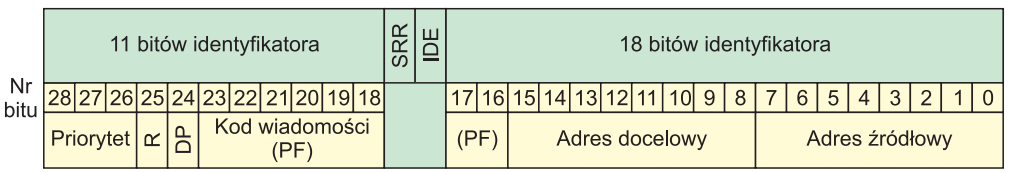
Gdy jedno urządzenie wysyła ramki danych, jedna po drugiej, wtedy może zaistnieć sytuacja, w której odbiorca danych nie nadąży z ich odbieraniem i przetwarzaniem. Wtedy z pomocą przychodzi ramka przepełnienia. Jej celem jest opóźnienie transmisji, zwolnienie przepływu informacji przez magistralę tak, aby przeciążone urządzenia mogły przetworzyć otrzymane dane. Postać ramki jest bardzo prosta, składa się jedynie z sześciu bitów dominujących oraz występujących po nich ośmiu bitach recesywnych. Wyżej umieszczono wzmiankę o mechanizmie umieszczania bitów komplementarnych w ramkach (*bit stuffing*), który w tym przypadku pośrednio zapewnia poprawną interpretację ramki przepełnienia. W normalnych warunkach nie może się pojawić po sobie więcej niż pięć bitów o tej samej wartości. Jeśli zostanie zarejestrowany dłuższy, ciąg wiadomo wtedy, że któreś urządzenie wysłało ramkę przepełnienia i nadajnik musi zwolnić wysyłanie danych.



Rys. 4. Budowa ramki danych w standardzie CAN 2.0B



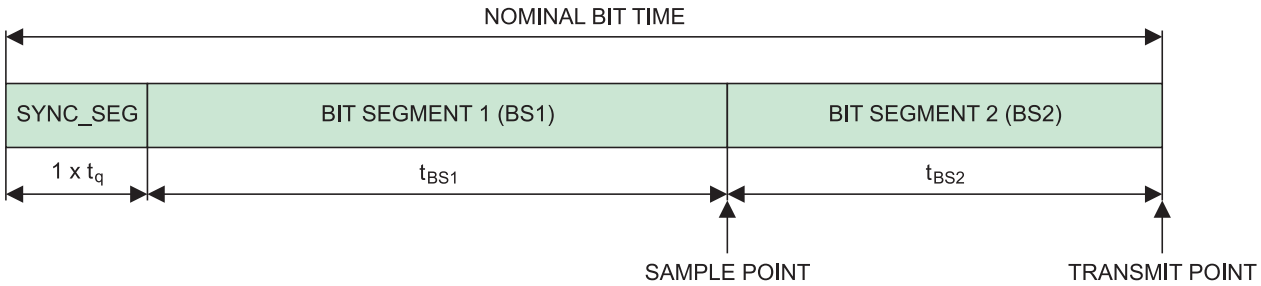
Rys. 5. Budowa ramki żądania transmisji



Rys. 6. Identyfikator wiadomości w standardzie J1939

Protokół transmisji

Protokołów zarządzających komunikacją po magistrali CAN jest co najmniej kilka. Są to m.in.: CAN-in-Automation, J1939. Wybór stosownego protokołu zależy przede wszystkim od tego, jakim zadaniom ma sprostać sieć CAN. Generalnie komunikacja może być prowadzona na dwa sposoby: zorientowana na wiadomości lub na obsługę adresów. Komunikacja oparta o wiadomości odbywa się na zasadzie wysyłania przez



Rys. 7. Cykl transmisji bitu dla zaimplementowanych w mikrokontrolerach STM32 kontrolerów CAN *(źródło: ST)*

List. 1. Funkcja *CAN_Configuration()*

```
void CAN_Configuration(void)
{
        CAN_DeInit(CAN1);
        CAN_StructInit(&CAN_InitStructure);

        CAN_InitStructure.CAN_TTCM = DISABLE;
        CAN_InitStructure.CAN_ABOM = DISABLE;
        CAN_InitStructure.CAN_AWUM = DISABLE;
        CAN_InitStructure.CAN_NART = DISABLE;
        CAN_InitStructure.CAN_RFLM = ENABLE;
        CAN_InitStructure.CAN_TXFP = DISABLE;
        CAN_InitStructure.CAN_Mode = CAN_Mode_Normal;
        CAN_InitStructure.CAN_SJW = CAN_SJW_1tq;
        CAN_InitStructure.CAN_BS1 = CAN_BS1_10tq;
        CAN_InitStructure.CAN_BS2 = CAN_BS2_7tq;
        CAN_InitStructure.CAN_Prescaler = 8;
        CAN_Init(CAN1, &CAN_InitStructure);

        CAN_FilterInitStructure.CAN_FilterNumber=0;
        CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;
        CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;
        CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;
        CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;
        CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;
        CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0000;
        CAN_FilterInitStructure.CAN_FilterFIFOAssignment=0;
        CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;
        CAN_FilterInit(&CAN_FilterInitStructure);
}
```

magistralę komunikatów z jakimś specyficznym identyfikatorem. Stacje podłączone do magistrali same decydują, czy wiadomość jest dla nich, czy nie. Drugi sposób komunikacji polega na przydzieleniu każdemu węzłowi sieci adresu, który jest kodowany w identyfikatorze wiadomości wysyłanej przez nadajnik. Takie przekazywanie informacji jest wykorzystywane m.in. w protokole transmisji J1939 opracowanym przez organizację SAE (*Society of Automotive Engineers*) dla pojazdów użytkowych.

Przykład budowy identyfikatora ramki danych w komunikacji zorientowanej na obsługę adresów z wykorzystaniem protokołu J1939 pokazano na **rys. 6**. Jest to identyfikator dla ramki w standardzie CAN 2.0B. Zaznaczono najbardziej interesujące miejsca: kod wiadomości, adres nadawcy oraz adres odbiorcy. Ponieważ długość każdego pola wynosi 8 bitów, liczba dostępnych adresów urządzeń wynosi 255, podobnie jak liczby rodzajów (kodów) wiadomości. W wielu przypadkach takie rozwiązanie może okazać się wystarczające. Prędkość transmisji zależy od odmiany protokołu J1939. W aplikacjach przedstawionych w dalszej części rozdziału wykorzystywana będzie namiastka protokołu w odmianie J1939/11, czyli o prędkości 250 kbit/s.

Kontroler CAN w mikrokontrolerach STM32

Kontroler magistrali CAN wbudowany w mikrokontrolery STM32 ma szereg dodatkowych możliwości, które pozytywnie wpływają na jakość pracy tych układów w sieciach CAN. Zarówno część odbiorcza kontrolera, jak i część nadawcza są wyposażone w sprzętowo zarządzane bufory mogące przechowywać odebrane (przeznaczone do wysłania) ramki, są to tzw. skrzynki pocztowe (*mailbox*).

Nadajnik ma zaimplementowane trzy skrzynki, o których kolejności wysyłania decyduje planista (*scheduler*). Dane przeznaczone do wysłania mogą być wpisywane tylko do pustych skrzynek. Po zapisaniu skrzynka ma nadawany status przyjęcia do realizacji i, w zależności od priorytetu, oczekuje na zwolnienie magistrali lub na wcześniejsze wysłanie zawartości skrzynek o wyższym priorytecie. Po zakończeniu wysyłania skrzynka uzyskuje status pustej i jest gotowa do ponownego zapisu.

Bufory odbiorcze (skrzynki) części odbiorczej są zorganizowane w kolejki FIFO. Poprawnie odebrane dane są zapisywane kolejno w poszczególnych skrzynkach. Jeśli wszystkie trzy skrzynki zostaną zapełnione danymi, a aplikacja na czas nie odczyta ich zawartości, to zachowanie kontrolera CAN, w zależności od ustawień, może być dwojakie. Nowo odebrana wiadomość może być odrzucona, wtedy w buforach przechowywane są trzy najstarsze wiadomości. Najświeższa wiadomość może też nadpisywać zawartość skrzynek, dzięki czemu przechowywane będą zawsze najnowsze wiadomości, a starsze będą tracone.

Komunikacja po magistrali CAN odbywa się z ustaloną prędkością, wyznaczaną przez czas trwania jednego bitu. Wyznaczenie cyklu transmisji jednego bitu (*bit timing*) nie jest tak oczywiste, jak ma to miejsce w większości szeregowych interfejsów komunikacyjnych. Ważnym pojęciem jest kwant czasu, ponieważ prędkość komunikacji jest determinowana przez jego wielokrotność. Kwant czasu to okres sygnału zegarowego magistrali PCLK1 podzielonego przez wartość preskalera kontrolera CAN.

Standard CAN definiuje cztery segmenty czasowe, na które podzielony jest czas trwania jednego bitu:

- segment synchronizacji *SYNC_SEG*,
- segment kompensacji czasu propagacji magistrali *PROP_SEG*,
- dwa segmenty wyznaczające miejsce próbkowania bitu – *PHASE_SEG1* oraz *PHASE_SEG2*.

Pierwsze pole czasowe bitu, czyli segment synchronizacji, trwa tyle, ile jeden kwant czasu. Wtedy to spodziewana jest ewentualna zmiana bitu. Dwa następne segmenty, czyli blok przeznaczony na kompensację propagacji magistrali (*Propagation Time Segment*) oraz segment wskazujący chwilę próbkowania *PHASE_SEG*, są często łączone w jeden blok *Bit segment 1* (*BS1*). Kontroler CAN w mikrokontrolerach STM32 również korzysta z modelu trójsegmentowego. Na **rys. 7** przedstawiono cykl transmisji bitu dla układów z rodziny STM32. Czas trwania segmentów BS1 oraz BS2 mogą ulegać automatycznej zmianie w celu kompensacji różnic częstotliwości pracy węzłów podłączonych do magistrali. Czas trwania segmentu BS1 jest programowalny i może wynosić od jednej do 16 wielokrotności kwantu czasu. Segment BS2 może trwać maksymalnie osiem kwantów czasu.

Prędkość komunikacji to odwrotność sumy czasów trwania wszystkich segmentów, można ją zatem wyznaczyć z zależności:

$$BR = \frac{1}{t_q + t_{BS1} + t_{BS2}}$$, gdzie: $$t_q = \frac{preskaler}{f_{PCLK1}}$$

Należy zaznaczyć, że wartość *preskaler* oznacza tutaj liczbę wpisaną do struktury inicjującej kontroler CAN, a nie wartość wpisywaną do rejestru preskalera. Żeby osiągnąć ten sam efekt bez użycia funkcji API firmy ST, należałoby do rejestru preskalera wpisać wartość *preskaler – 1*. Sposób wyznaczania prędkości komunikacji stanie się bardziej zrozumiały po zapoznaniu się z poniższym przykładem. Celem jest osiągnięcie prędkości 250 kbit/s.

List. 2. Struktury ramek: nadawczej i odbiorczej

```
typedef struct
{
        uint32_t StdId;
        uint32_t ExtId;
        uint8_t IDE;
        uint8_t RTR;
        uint8_t DLC;
        uint8_t Data[8];
} CanTxMsg;


typedef struct
{
        uint32_t StdId;
        uint32_t ExtId;
        uint8_t IDE;
        uint8_t RTR;
        uint8_t DLC;
        uint8_t Data[8];
        uint8_t FMI;
} CanRxMsg;
```

Częstotliwość PCLK1 wynosi zazwyczaj 36 MHz. Przyjmując czasy trwania segmentów BS1 i BS2 odpowiednio: 10 i 7 wielokrotności kwantu czasu oraz *preskaler = 8*, otrzymujemy:

$$BR = \frac{f_{PCLK1}}{18 \cdot preskaler} = \frac{36[MHz]}{144} = 250[kbit/s]$$

Filtry akceptacyjne

Magistrala CAN jest siecią rozgłoszeniową, a więc na każdej stacji podłączonej do magistrali ciąży obowiązek filtrowania nadchodzących ramek i decydowania, czy otrzymane dane są przeznaczone dla niej, czy nie. Można do rozwiązania tego problemu podejść od strony programowej, jednak obciążenie mikroprocesora w tym przypadku nie będzie niezauważalne. Znacznie lepszym pomysłem jest filtrowanie ramek już na poziomie sprzętowym. Zadanie to jest wykonywane przez filtry akceptacyjne. Filtry można tak ustawić, aby do buforów odbiorczych były przepisywane jedynie ramki, które są przeznaczone dla danego węzła (np. mają zakodowany odpowiedni adres odbiorcy). Zyskiem z używania filtrów sprzętowych jest znaczne odciążenie rdzenia układu mikroprocesorowego, ponieważ obsługiwane są jedynie wiadomości przeznaczone dla danego urządzenia, reszta jest po prostu pomijana.

Aplikacja

Omawiana aplikacja przykładowa może być w łatwy sposób przenoszona pomiędzy różnymi zestawami uruchomieniowymi wyposażonymi w mikrokontroler STM32 i układ nadajnika/odbiornika CAN. Przedstawiona konfiguracja dotyczy zestawu ewaluacyjnego STM3210B-EVAL z mikrokontrolerem STM32F103 oraz w transceiverem SN65HVD230, podłączonym do mikrokontrolera według schematu przedstawionego na **rys. 8**. Aplikację przygotowano w pakiecie TrueSTUDIO firmy Atollic. W zastosowanym zestawie ewaluacyjnym transceiver podłączono do alternatywnych wyprowadzeń kontrolera CAN, a zatem do poprawnej pracy wymagane jest przemapowanie wyprowadzeń.

Po standardowej konfiguracji sygnałów zegarowych niezbędną czynnością potrzebną do poprawnej pracy układu kontrolera CAN, podobnie jak ma to miejsce dla wszystkich układów peryferyjnych wbudowanych w mikrokontrolery STM32, jest włączenie sygnału taktującego. Ta oraz wszystkie pozostałe czynności konfiguracyjne związane z kontrolerem CAN są przeprowadzane w wywołaniu funkcji *CAN_Configuration()*, której ciało zamieszczono na **list. 1**. Przedstawiony fragment kodu jest wyraźnie podzielony na dwa bloki. Pierwszy odpowiada za konfigurację samego sterownika CAN, natomiast drugi wprowadza parametry pracy sprzętowego filtru akceptacyjnego ramek przychodzących.

Poprzez wypełnianie kolejnych pól struktury *CAN_InitStructure* ustalane są parametry pracy magistrali. Wyłączane są;

- komunikacja wyzwalana czasowo (*TTCM – Time Triggered Communication Mode*),
- automatyczne zarządzanie odcinaniem od sieci (*ABOM – Automatic Bus-off Managment*),
- trybu automatycznego wybudzania (*AWUM – Automatic Wake-up Mode*),
- priorytety kolejek FIFO dla danych wysyłanych (TXFP – *Transmit FIFO Priority*),

Natomiast włączane są:

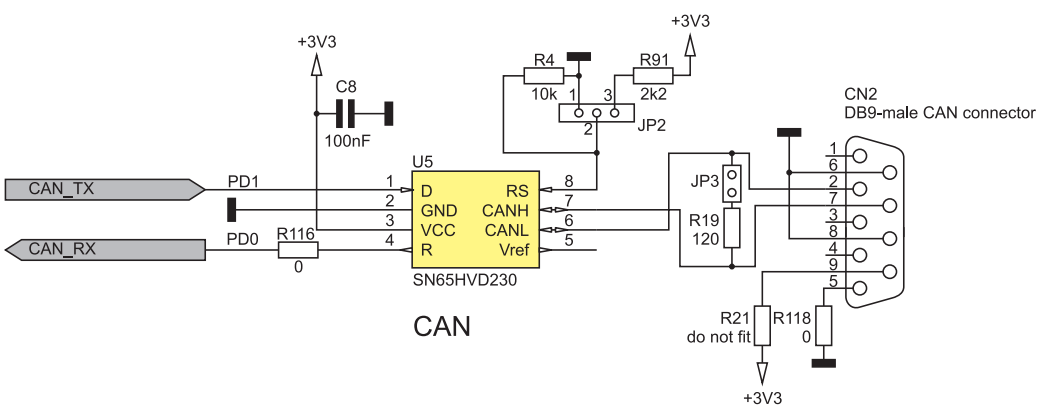
- tryb automatycznej retransmisji (*NART – No-Automatic Retransmission Mode*), tutaj warto zaznaczyć, że występuje podwójne negowanie – wyłączany jest tryb nieautomatycznej retransmisji danych,
- tryb zabezpieczenia przed nadpisaniem danych przechowywanych w kolejkach FIFO przez nowe dane (*RFLM – Receive FIFO Locked Mode*).

Dalej wybierany jest tryb pracy kontrolera CAN. Tutaj wybrano tryb normalny, ale istnieją jeszcze trzy inne tryby pracy: w pętli powrotnej, cichy i połączenie obydwu. Praca w trybie normalnym oznacza pełny dostęp do magistrali. Tryb pętli powrotnej polega na odcięciu odbiornika CAN od magistrali i podłączeniu w to miejsce bezpośrednio części nadawczej CAN. W konsekwencji sterownik nie może odbierać danych z magistrali. Analogicznie praca w trybie cichym (*silent mode*) oznacza niemożność wysyłania danych przez magistralę. Kombinacja obydwu powyższych wiąże się z całkowitym odcięciem od magistrali, a dane są wewnętrznie kierowane z nadajnika CAN do odbiornika.

W kolejnym kroku konfiguracji ustalane są czasy trwania segmentów cyklu transmisji bitu, a więc pośrednio również prędkość komunikacji. Ponieważ wartość preskalera wynosi osiem, prędkość transmisji dla takich ustawień będzie wynosić 250 kbit/s.

Ramki – nadawcza oraz odbiorcza – mają w bibliotece firmy ST specjalnie dla nich stworzony typ danych. Są to struktury, których deklaracje zamieszono na **list. 2**. Wysłanie ramki danych odbywa się za pomocą wywołania funkcji *CAN_Transmit()*. Odbiór, jeśli włączone zostało przerwanie *USB_LP_CAN1_RX0_IRQn*, może zostać przeprowadzony w funkcji obsługi przerwania *USB_LP_CAN1_RX0_IRQHandler*, która powinna zostać zamieszczona w pliku *stm32f10x_it.c*.

Krzysztof Paprocki, EP
krzysztof.paprocki@ep.com.pl



Rys. 8. Schemat połączeń pomiędzy mikrokontrolerem a układem transceivera CAN *(źródło ST)*