Urządzenie diagnostyczne do sieci CAN

AVT 5280

Analiza ruchu sieciowego umożliwia realizację szeregu zadań diagnostycznych pozwalających na np. wykrywanie różnego rodzaju nieprawidłowości, symulowanie nieprzewidzianych sytuacji, optymalizowanie sieci, przeprowadzanie konfiguracji, zbieranie danych statystycznych itp. Wymaga to nie tylko dużej wiedzy, ale również odpowiednich narzędzi sprzętowych i programistycznych. Jednymi z nich są przyrządy diagnostyczne umożliwiające „nasłuchiwanie" pakietów danych.

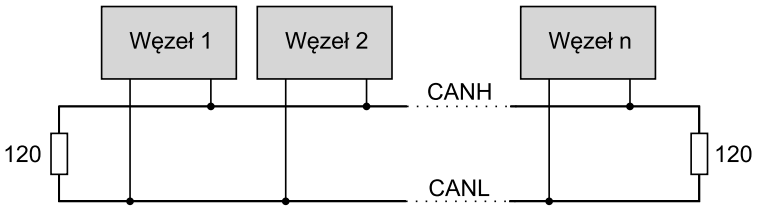
***Rekomendacje:** przyrząd przyda się integratorom systemów automatyki przemysłowej, serwisom motoryzacyjnym oraz konstruktorom urządzeń elektronicznych przeznaczonych dla motoryzacji.*

CAN jest standardem komunikacyjnym powstałym w latach osiemdziesiątych XX wieku w firmie Robert Bosch GmbH z myślą o zastosowaniach w przemyśle samochodowym. Entuzjastyczne przyjęcie tej technologii przez motoryzację zaowocowało szybką ekspansją standardu do innych dziedzin wymagających przesyłu danych. Obecnie CAN jest popularną przemysłową metodą transmisji danych, wykorzystywaną w różnego rodzaju zastosowaniach cywilnych i wojskowych. Są to nie tylko środki transportu (samochody osobowe i ciężarowe, autobusy, samoloty, statki, maszyny rolnicze itp.), ale również np. systemy automatyki budynkowej, sieci sensorowe i wiele innych.

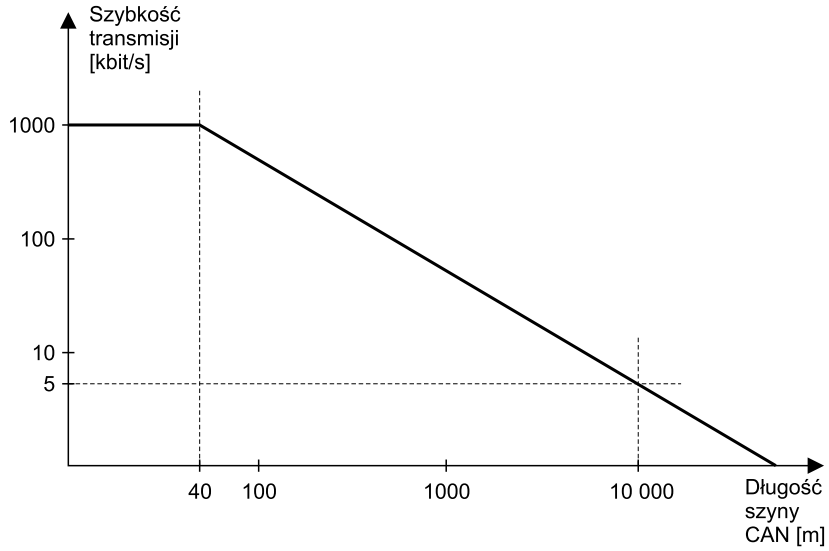
Jest to standard zdefiniowany w oparciu o model warstwowy OSI/ISO. Specyfikacja CAN definiuje dwie pierwsze warstwy: fizyczną i łącza danych. Warstwy 3...6. nie są używane, natomiast warstwę 7. (aplikacji) pozostawiono do zdefiniowania przez projektantów danego systemu. CAN w warstwie 1. jest zbudowany z dwóch przewodów nazywanych *CAN High* (CANH) oraz *CAN Low* (CANL), które służą do różnicowego przesyłania sygnałów (stan magistrali i dane interpretowane są za pomocą różnicy napięć między przewodami). Linie sygnałowe na obu końcach są ze sobą połączone za pomocą rezystorów o oporności 120 Ω, co zapewnia dopasowanie energetyczne. Podstawową topologią sieci CAN jest magistrala, a więc architektura sieciowa polegająca na połączeniu szeregowym węzłów sieci do wspólnego medium transmisyjnego (**rysunek 1**).

W sieci CAN można przesyłać dane z prędkością do 1 Mbit/s, jednak taką wartość prędkości można uzyskać tylko w magistralach o małym zasięgu, nieprzekraczającym 40 metrów. Wydłużenie magistrali jest możliwe do uzyskania kosztem prędkości transmisji. W skrajnym przypadku można osiągnąć długość magistrali rzędu 10 km (przy prędkości transmisji 5 kbit/s, **rysunek 2**).

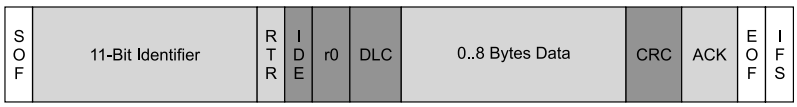
Warstwa druga modelu OSI/ISO standardu CAN specyfikuje cztery rodzaje ramek komunikacyjnych. Są to: ramka danych, ramka zdalnego wywołania, ramka sygnalizacji błędu, ramka przepełnienia. Informacje są przesyłane w ramce danych. Dokumentacja firmy Bosch wyróżnia dwa rodzaje ramek danych: standardową (wersja CAN 2.0A) oraz rozszerzoną (CAN 2.0B). Te dwie wersje definiują odmienne formaty ramek wiadomości, głównie różniące się długością identyfikatora. Standard CAN 2.0A cechuje się adresem ramki o długości 11 bitów (**rysunek 3**), natomiast w standardzie CAN



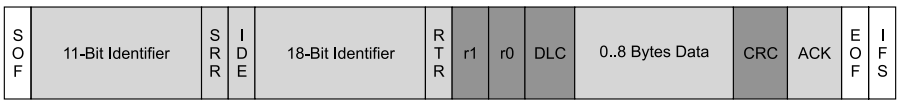
Rysunek 1. Struktura sieci CAN



Rysunek 2. Korelacja pomiędzy szybkością przesyłania danych a długością magistrali



Rysunek 3. Struktura ramki danych (format ramki standardowej – CAN 2.0A)



Rysunek 4. Struktura ramki danych (format ramki rozszerzonej – CAN 2.0B)

2.0B adres zawiera 29 bitów (**rysunek 4**). Niezależnie od używanego standardu wielkość pola danych jest niezmienna i wynosi 8 bajtów. Pozostałe elementy składające się na ramkę danych są wykorzystywane jako pola sterujące, konfiguracyjne lub zapewniające poprawność przesyłanych danych:

- SOF (*Start Of Frame*) – początek ramki,
- RTR (*Remote Transmission Request Bit*) – rodzaj ramki,
- IDE (*Identifier Extension*) – format ramki danych (podstawowy lub rozszerzony),
- r0 – bit zarezerwowany,
- DLC (*Data Length Code*) – liczba bajtów danych,
- CRC (*Cyclic Redundancy Check)* – suma kontrolna,
- ACK (Acknowledgement) – potwierdzenie wysłania/odebrania danych,
- EOF (*End of Frame*) – zakończenie ramki,
- IFS (*Intermission*) – przerwa przed następną ramką.

CAN jest standardem transmisyjnym typu multi-master (nie ma wyodrębnionej jednej, niezmiennej jednostki nadrzędnej), dlatego konieczna stała się implementacja algorytmów regulujących dostęp do medium transmisyjnego w taki sposób, by nie dopuścić do utraty danych przy próbie dostępu do medium więcej niż jednego modułu w danej chwili. Zastosowano metodę dostępu do medium zwaną CSMA, umożliwiającą śledzenie stanu dostępności medium i wykrywanie kolizji w przypadku próby równoczesnej transmisji danych przez więcej niż jeden moduł.

Brak zdefiniowania warstwy 7. stwarza możliwość jej implementacji zgodnie z indywidualnymi zapotrzebowaniami i konkretnymi zastosowaniami danej sieci. Na przestrzeni lat powstało wiele protokołów bazujących na standardzie CAN z określoną warstwą aplikacyjną. Główne z nich wymieniono w **tabeli 1**. Znajdują one zastosowanie w różnorodnych dziedzinach przemysłu.

Rysunek 5. Interpretacja funkcjonalności urządzenia

AVT-5280 w ofercie AVT:
AVT-5280A – płytka drukowana
AVT-5280B – płytka drukowana + elementy

Podstawowe informacje:
- Płytka o wymiarach 39 mm×34 mm
- Zasilanie z portu USB
- Komunikacja USB w trybie Virtual COM Port
- Układ mikroprocesorowy ARM Cortex-M3 z rodziny STM32 (STM32F105RCT6) firmy ST Microelectronics
- Nadawanie i odczyt ramek CAN ze sterowaniem z poziomu komputera PC

Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 10460, pass: 0646g3n0
- wzory płytek PCB
- karty katalogowe i noty aplikacyjne elementów oznaczonych w Wykazie elementów kolorem czerwonym

Tabela. 1. Protokoły warstwy 7. oparte na standardzie CAN

Nazwa	Główne obszary wykorzystania
NMEA 2000	Statki
J1939	Pojazdy kołowe
DeviceNet	Zastosowanie przemysłowe np. fabryki
CANopen	Systemy wbudowane
ISOBUS	Maszyny rolnicze
MILCAN	Zastosowania wojskowe

Koncepcja budowy i dobór podzespołów

Obecnie w ofercie producentów znajdują się liczne urządzenia diagnostyczne do sieci CAN. Analiza ich funkcjonalności umożliwia określenie podstawowych cech autorskiej konstrukcji, która w możliwie dużym stopniu powinna odzwierciedlać profesjonalne, ale zarazem kosztowne narzędzia komercyjne. Głównym zadaniem tego typu urządzeń jest zapewnienie dostępu do sieci CAN, co polega na:

- odbiorze ramek CAN przesyłanych przez sieć i wizualizowanie ich operatorowi za pomocą czytelnego interfejsu,

R E K L A M A

Na CD: karty katalogowe i noty aplikacyjne elementów oznaczonych w Wykazie elementów kolorem czerwonym

Wykaz elementów

Rezystory:
R1: 10 kΩ (SMD, 0805)
R2: 1 MΩ (SMD, 0805)
R3: 1,5 kΩ (SMD, 0805)
R4, R5: 22 Ω (SMD, 0805)
R6: 120 Ω (SMD, 0805)
R7...R9: 150 Ω (SMD, 0805)
Kondensatory:
C1..C6: 100 nF (SMD, 0805)
C7, C8: 22 pF (SMD, 0805)
C9: 10 µF (tantalowy, A)
C10, C11: 1 µF (0805)
Półprzewodniki:
IC1: STM32F103RCT6
IC2: MCP1802T-33
IC3: SN65HVD230
Inne:
Q1: 25 MHz (SMD)
RX, TX, PWR: dioda LED (3 mm)
USB_CON: gniazdo USB typu B, kątowe
SWD_CON: gniazdo IDC10 proste
JP1: złącze szpilkowe (2 goldpiny)
JP2: złącze szpilkowe (3 goldpiny)
Obudowa ABS-2A

Tab. 3. Najważniejsze dane techniczne mikrokontrolera STM32F105RCT6

Nazwa parametru	Wartość
Napięcie zasilania	2.0…3.6 V
Częstotliwość taktowania	do 72 MHz
Ilość pamięci FLASH	256 kB
Ilość pamięci SRAM	64 kB
Ilość i rodzaj liczników	cztery 16-bitowe ogólnego przeznaczenia, jeden 16-bitowy dedykowany do sterowania silnikiem prądu stałego, dwa 16-bitowe do przetwornika C/A, dwa typu Watchdog, jeden 24-bitowy systemowy
Liczba i rodzaj przetworników A/C	dwa 12-bitowe
Liczba i rodzaj przetworników C/A	dwa 12-bitowe
Rodzaj i ilość interfejsów komunikacyjnych	USB (1), CAN (2), USART (5), I²C (2), SPI (3)
Temperatura pracy	–40…+ 125 °C
Obudowa	LQFP64

- wysyłaniu ramek CAN o parametrach pożądanych przez operatora.

Spełnienie tych założeń realizowane jest poprzez dwukierunkową konwersję CAN/USB, dzięki czemu cała warstwa interfejsu użytkownika dostępna jest w postaci aplikacji komputerowej. Blokowy model funkcjonalności urządzenia przedstawiono na **rysunku 5**.

Przyjęcie takiej koncepcji wymaga wyboru odpowiednich układów elektronicznych. Należy do nich przede wszystkim mikrokontroler, który powinien być wyposażony w interfejsy CAN oraz USB. Następnym potrzebnym elementem jest układ transmisyjny CAN umożliwiający komunikację z sieciami pracującymi w tym standardzie.

Stacjonarny charakter urządzenia i obsługa transmisji USB umożliwiają zasilanie urządzenia z gniazda USB, dlatego kolejną integralną częścią testera powinien być stabilizator napięcia. Schemat blokowy obrazujący wzajemne relacje między tymi komponentami przedstawiono na **rysunku 6**.

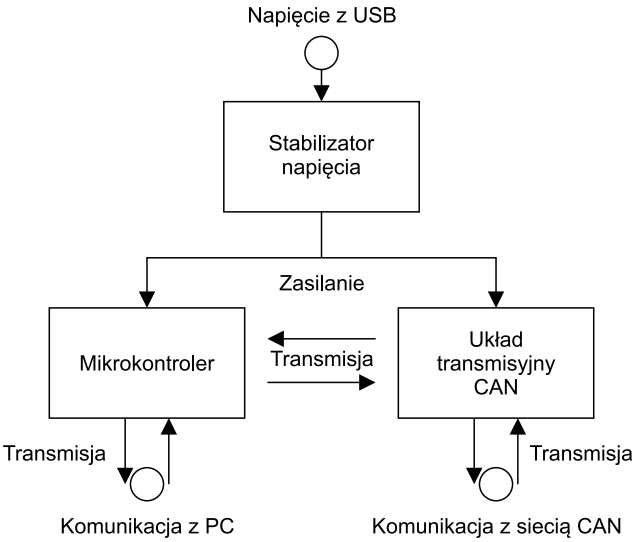
W zrealizowanym projekcie wykorzystano mikrokontroler STM32F105RCT6 firmy STMicroelectronics. Należy on do rodziny układów STM32 (rdzeń ARM Cortex-M3) z grupy Connectivity Line. Mikrokontroler jest bogato wyposażony w zasoby wewnętrzne, do których należą duża ilość pamięci (256 kB Flash, 64 kB SRAM) oraz liczne peryferie: moduły zegarowe (PLL do 72 MHz), układy licznikowe, przetwornik A/C oraz C/A, interfejsy komunikacyjne. Wśród ostatniej wymienionej grupy znajdują się interfejsy CAN (2) oraz USB (1). Istotną cechą jest możliwość równoczesnego korzystania z obu peryferii, co było niemożliwe w dotychczasowych układach STM32, gdzie interfejsy te korzystają ze współdzielonych zasobów. Szczegółowe zestawienie parametrów układu STM32F105RCT6 przedstawiono w **tabeli 3**.

Jako układ nadawczo-odbiorczy CAN użyto SN65HVD230 firmy Texas Instruments. Do komunikacji z mikrokontrolerem układ ma wyprowadzenia D i R, natomiast CANH i CANL do połączenia z magistralą CAN (**rysunek 7**). Do cech charakterystycznych układu można zaliczyć niski pobór prądu, mechanizm wyłączenia w przypadku przegrzania oraz dostosowanie do zasilania napięciem 3,3 V (**tabela 4**).

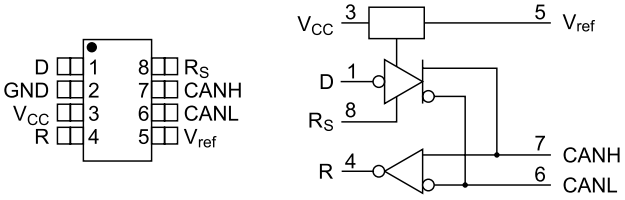
Na potrzeby projektu zastosowano stabilizator MCP1802T-33, który służy do zasilania mikrokontrolera oraz układu interfejsowego CAN. Atutem tego układu jest niewielka liczba komponentów potrzebnych do prawidłowej pracy wystarczą dwa kondensatory ceramiczne 1 µF. Wyprowadzenia oraz strukturę wewnętrzną układu widać na **rysunku 8**. Dokładne parametry stabilizatora zamieszczono w **tabeli 5**.

Budowa

Schemat ideowy urządzenia pokazano na **rysunku 9**. W centralnym miejscu umieszczono mikrokontroler STM-



Rysunek 6. Schemat blokowy analizatora CAN



Rysunek 7. Wyprowadzenia oraz struktura wewnętrzna układu SN65HVD230

Tab. 4 Parametry układu SN65HVD230

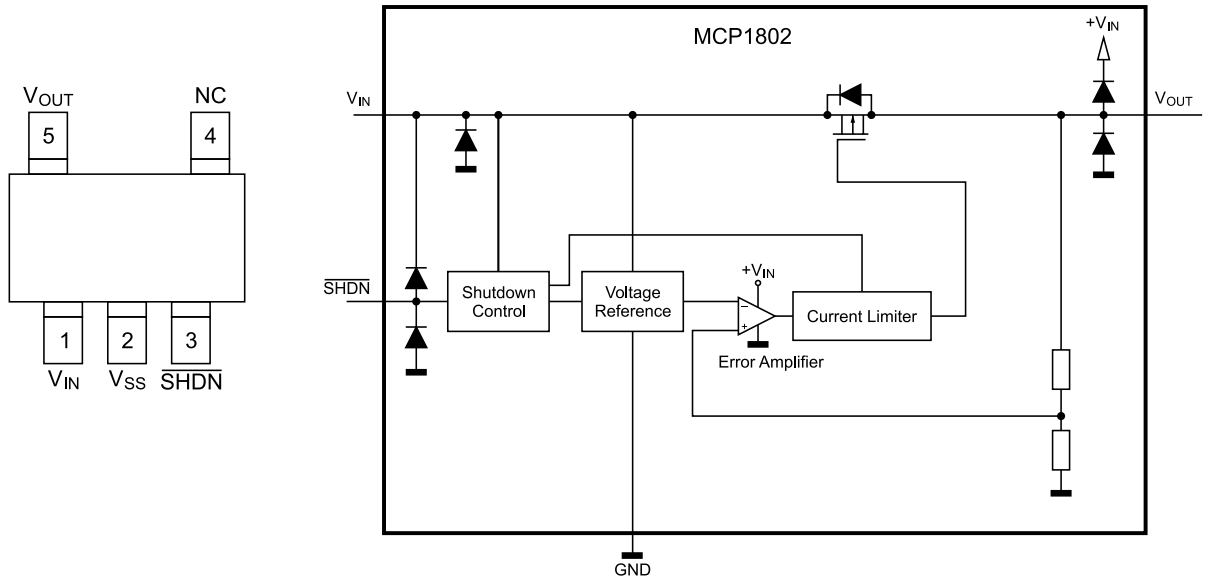
Nazwa parametru	Wartość
Napięcie zasilania	3.0…3.6 V
Prędkość transmisji	do 1 Mbit/s
Pobór prądu Tryb Normal Tryb Sleep mode Tryb Standby mode	 do 17 mA 40 nA 370 uA
Temperatura pracy	–40…+ 85 °C
Obudowa	8SOIC

32F105RCT6 (IC1). Jego schemat aplikacyjny zaczerpnięty został z dokumentacji producenta. Zgodnie z zaleceniami producenta do każdego wyprowadzenia zasilającego (VBAT, VDDA, VDD1…VDD4) dołączono ceramiczny kondensator blokujący 100 nF (C1...C6).

W celu wybrania rodzaju pamięci, z której wykonywany jest kod programu,

Tab. 5 Parametry układu MCP1802T-33

Nazwa parametru	Wartość
Napięcie wyjściowe	3,3 V
Maksymalne napięcie wejściowe	10 V
Poziom napięcia dropout	200mV @ 100 mA
Wydajność prądowa	300 mA
Prąd upływności	25 uA
Pobór prądu trybie Shutdown	0.01 uA
Dokładność napięcia wyjściowego	± 2%
PSRR	70 dB @ 10kHz
Obudowa	SOT-23-5



Rysunek 8. Wyprowadzenia oraz struktura wewnętrzna układu MCP1802T-33

nóżka BOOT0 połączona została z masą, co powoduje wybranie wewnętrznej pamięci Flash układu.

Kolejnym dołączonym do mikrokontrolera elementem jest kwarc Q1. Służy on jako źródło sygnału zegarowego, umożliwiając taktowanie układu z wykorzystaniem pętli PLL. Elementami wspierającymi działanie kwarcu są kondensatory C7, C8 oraz rezystor R2.

Programowanie i debugowanie mikrokontrolera odbywa się poprzez interfejs SWD, którego linie wyprowadzono na gnieździe SWD_CON. Standard ten ma liczne zalety, gdyż wymaga minimalnej liczby połączeń oraz nie potrzebuje dodatkowych rezystorów podciągających, zachowując jednocześnie pełną funkcjonalność klasycznego interfejsu JTAG.

Linie sygnałowe USB D+ oraz USB D– są doprowadzone do mikrokontrolera przez włączone szeregowo rezystory R4, R5 (odpowiednio PA12, PA11). Dodatkowo, do linii USB D+ dołączono rezystor podciągający R3, którego obecność informuje kontroler USB komputera o zgodności z trybami transmisji *USB Full Speed* oraz *USB High Speed*.

Pierwszym układem peryferyjnym mikrokontrolera jest transceiver CAN (IC3). Transmisja między nim a mikrokontrolerem odbywa się za pomocą linii sygnałowych TXD i RXD. Połączenie układu z magistralą CAN jest możliwe przez złącze JP2, gdzie wyprowadzone zostały linie CAN High oraz CAN Low. Równolegle między obiema liniami został umieszczony rezystor R6. Jego dołączenie do magistrali jest możliwe przez zamknięcie zworki JP1, co powinno zostać zrobione, gdy analizator jest dołączony na jednym z końców magistrali CAN.

Napięcie wejściowe stabilizatora IC2 pochodzi z gniazda USB (5 V), po czym jest ono filtrowane przez kondensator tantalowy C8. Dodatkowo zgodnie ze schematem aplikacyjnym producenta, na wejściu i wyjściu układu umieszczono po jednym kondensatorze ceramicznym o pojemności 1 µF (C10, C11). Stabilizator zapewnia napięcie 3,3 V i jest źródłem zasilania dla mikrokontrolera oraz układu transmisyjnego CAN.

Stan pracy urządzenia sygnalizowany jest przez diody LED. Dioda PWR (kolor zielony) informuje o obecności napięcia zasilania. Diody TX (kolor żółty) i RX (kolor czerwony) informują o trwającej w danej chwili transmisji (odpowiednio nadawczej dla TX i odbiorczej dla RX).

Listing 1. Realizacja konwersji CAN=>USB

```
void CAN1_RX0_IRQHandler(void)
{
    CAN_Receive(CAN1, CAN_FIFO0, &RxMessage); //odbior ramki CAN
    if(RxMessage.ExtId != 0)                  //adres (rozszerzony)
    {
           RX_Buffer[0] = (RxMessage.ExtId & 0xFF000000)>>24;
        RX_Buffer[1] = (RxMessage.ExtId & 0x00FF0000)>>16;
        RX_Buffer[2] = (RxMessage.ExtId & 0x0000FF00)>>8;
        RX_Buffer[3] =  RxMessage.ExtId & 0x000000FF;
    }

    if(RxMessage.StdId != 0)                  //adres (podstawowy)
    {
           RX_Buffer[0] = (RxMessage.StdId & 0xFF000000)>>24;
        RX_Buffer[1] = (RxMessage.StdId & 0x00FF0000)>>16;
        RX_Buffer[2] = (RxMessage.StdId & 0x0000FF00)>>8;
        RX_Buffer[3] =  RxMessage.StdId & 0x000000FF;
    }

    RX_Buffer[4] = RxMessage.Data[0];    //dane
    RX_Buffer[5] = RxMessage.Data[1];
    RX_Buffer[6] = RxMessage.Data[2];
    RX_Buffer[7] = RxMessage.Data[3];
    RX_Buffer[8] = RxMessage.Data[4];
    RX_Buffer[9] = RxMessage.Data[5];
    RX_Buffer[10] = RxMessage.Data[6];
    RX_Buffer[11] = RxMessage.Data[7];
    RX_Buffer[12] = RxMessage.DLC;       //dlugosc pola danych
    RX_Buffer[13] = RxMessage.IDE;       //rodzaj ramki

    USB_SIL_Write(EP1_IN,RX_Buffer,14);  //transfer USB
    SetEPTxValid(ENDP1);

    GPIO_SetBits(GPIOA, GPIO_Pin_1);     //miganie diody LED
    my_delay(20000);
    GPIO_ResetBits(GPIOA, GPIO_Pin_1);
}
```

Zasada działania i obsługa

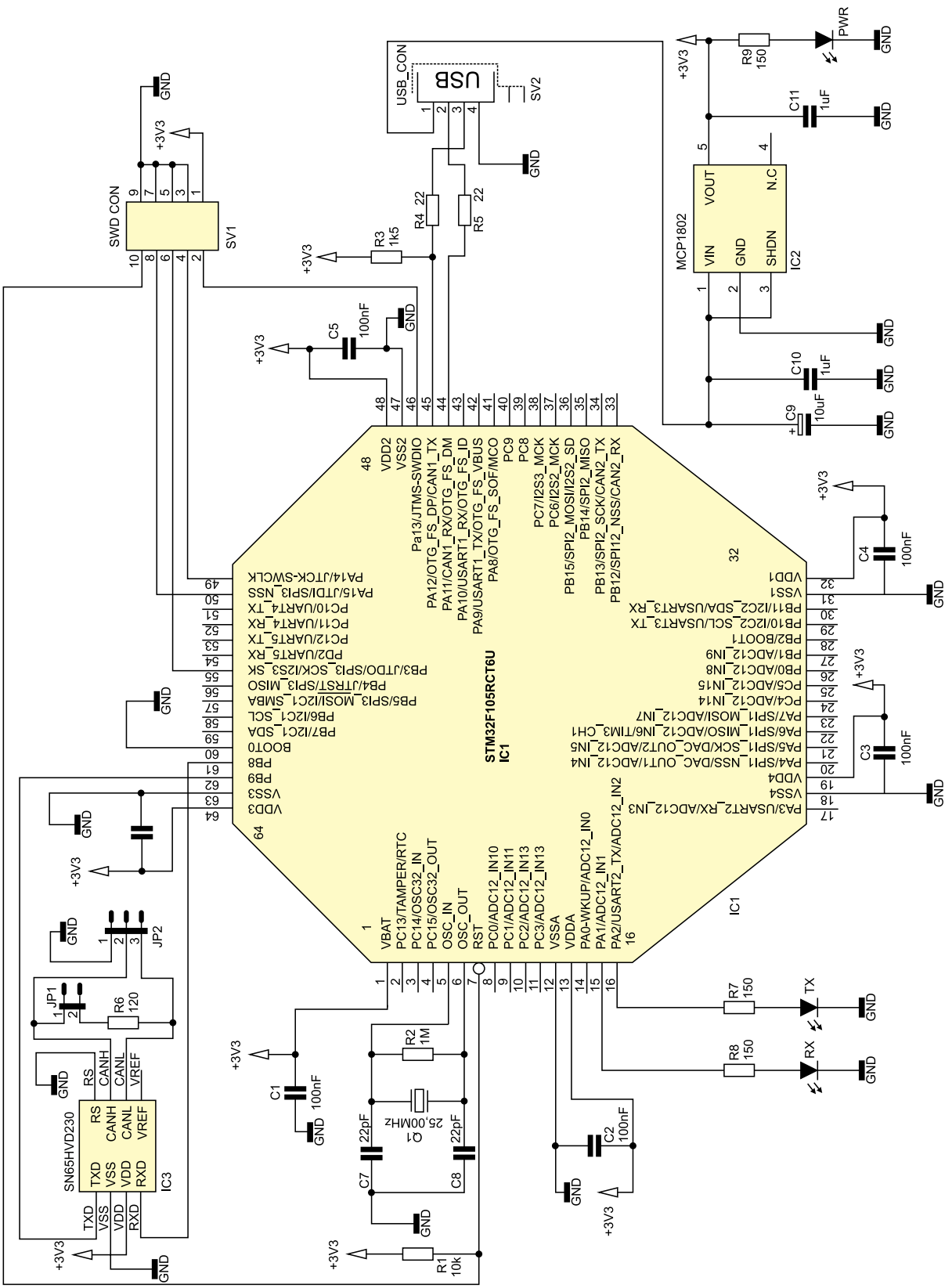
Schemat blokowy programu mikrokontrolera przedstawiono na **rysunku 10**. Po dołączeniu analizatora CAN do gniazda USB komputera PC następuje automatyczne włączenie urządzenia. Działanie programu rozpoczyna się od skonfigurowania układu

R E K L A M A

STM32. Na początku włączony zostaje zegar taktujący rdzeń (zwielokrotniony przez pętlę PLL) oraz zegary dla peryferii (CAN, USB). Następnie zostają skonfigurowane wyprowadzenia: PA1, PA2 (diody LED), PA11, PA12 (USB), PB8, PB9 (CAN). Kolejnym krokiem jest inicjalizacja interfejsów CAN oraz USB. Oba zostają uruchomione z obsługą przerwań. Po wykonaniu tej czynności kończy się etap konfigurowania urządzenia.

Sposób dołączenia urządzenia do magistrali CAN pokazano na **rysunku 11.** Będąc jednym z węzłów sieci, analizator ma możliwość odbioru wszystkich ramek. Każdorazowo po odebraniu ramki odczytuje ją, kopiuje z niej informacje o adresie, danych, rodzaju ramki i długości pola danych, po czym przesyła je do komputera interfejsem USB. Realizującą te czynności funkcję obsługi przerwania *CAN1_RX0_IRQHandler(void)* przedstawiono na **listingu 1**.

Wewnątrz niej znajduje się funkcja odbierająca ramki CAN *CAN_Receive(CAN1, CAN_FIFO0, &RxMessage)*. Odczytana ramka znajduje się w strukturze *RxMessage*, której pola z potrzebnymi informacjami kopio-



Rysunek 9. Schemat elektryczny analizatora CAN

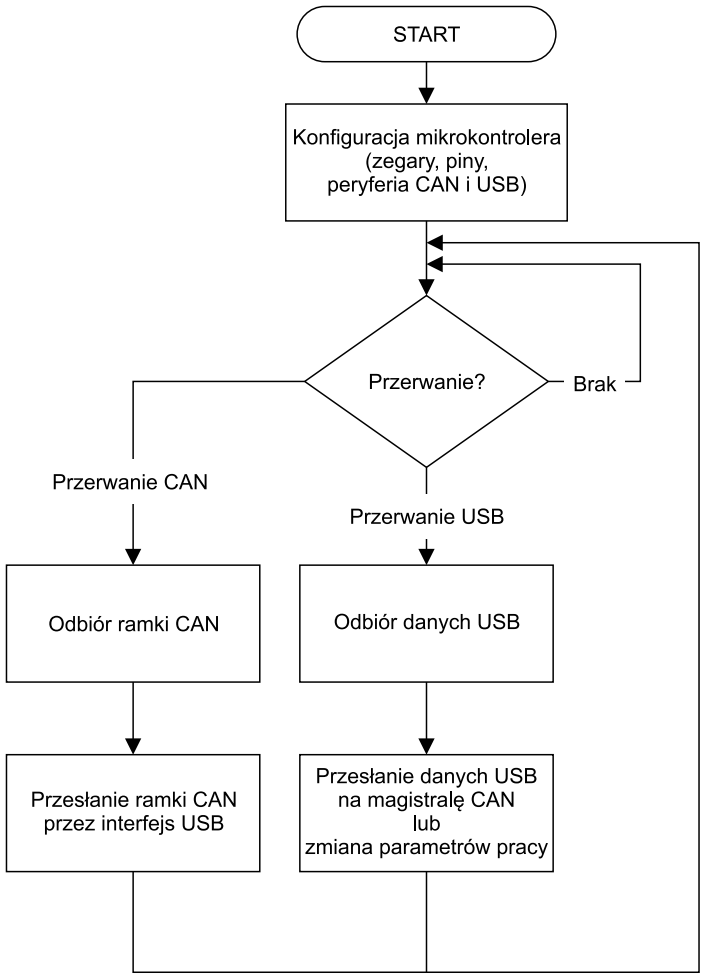
wane są do 14-bajtowego bufora *RX_Buffer*, a następnie przesyłane do komputera funkcją *USB_SIL_Write(EP1_IN,RX_Buffer,14)*.

Konwersja w odwrotną stronę przebiega analogicznie. Odpowiednio sformatowane dane są przesyłane interfejsem USB i odbierane przez mikrokontroler (funkcja *USB_SIL_Read(EP3_OUT, USB_Rx_Buffer)*). Następnie tworzona jest nowa ramka CAN, wypełniane są pola: rodzaju ramki, adresu, danych itp., po czym gotowa ramka wysyłana jest na magistralę (funkcja *CAN_Transmit(CAN1,&TxMessage)*).

Wizualizacja odbieranych przez komputer ramek z sieci CAN oraz możliwość wysyłania ramek w tym standardzie wymagają stworzenia przeznaczonej do tego celu aplikacji komputerowej, która udostępniać będzie operatorowi interfejs użytkownika. Znacznym ułatwieniem tego zadania jest wykorzystywany przez mikrokontroler STM32 tryb pracy interfejsu USB, którym jest VCP (*Virtual COM Port*). Tryb ten tworzy w komputerze wirtualny interfejs szeregowy COM (**rysunek 11**, **rysunek 12**). Dzięki pracy interfejsu USB w trybie VCP do obsługi analizatora CAN zamiast skomplikowanej implementacji stosu USB wystarczy prosta aplikacja wykorzystująca popularny interfejs RS-232.

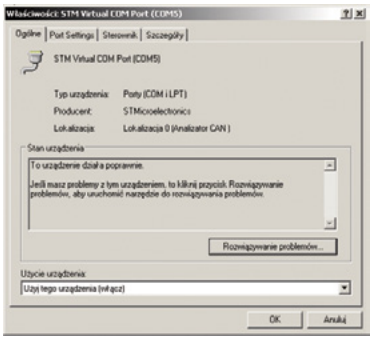
Do napisania aplikacji komputerowej wybrano język C# i środowisko Microsoft Visual C#. Platforma ta ma wbudowane komponenty programowe do obsługi transmisji RS-232, dzięki czemu stworzenie odpowiedniego nie jest trudne (**rysunek 13**).

Widok gotowej aplikacji przedstawiono na **rysunku 14**. Wyróżnić w niej można cztery bloki funkcjonalne. Pierwszy (prawy górny róg) służy do nawiązywania komunikacji z analizatorem CAN, co realizowane jest przez wybranie i połączenie się z odpowiednim portem COM. Drugi blok (prawy dolny róg) umożliwia zmianę konfiguracji pracy analizatora CAN (przełączenie na inną prędkość transmisji CAN). Kolejny blok (lewy dolny róg) realizuje wysyłanie ramek CAN. Ostatni z bloków (lewy górny róg) to pole wyświetlające odebrane ramki



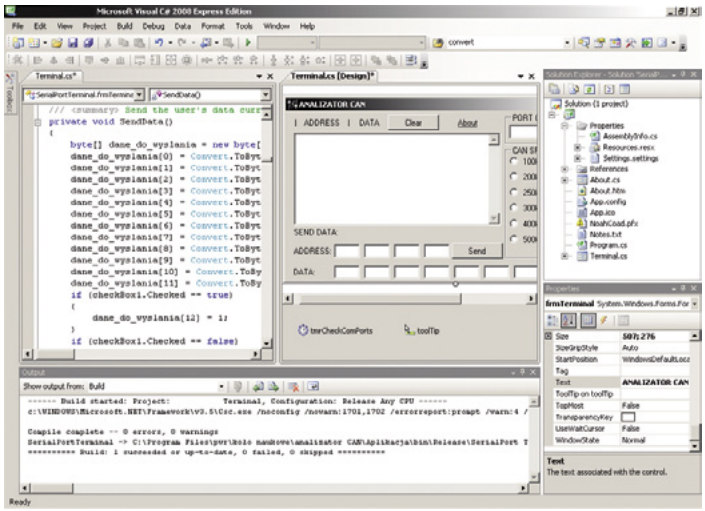
Rysunek 10. Schemat blokowy oprogramowania

Rysunek 11. Widok zakładki menedżera urządzeń w systemie Windows (analizator CAN widoczny jako STM Virtual COM Port)



Rysunek 12. Informacje na temat analizatora CAN widoczne z poziomu systemu Windows (zakładka „Właściwości" w menedżerze urządzeń)

CAN. Każda linia to jedna ramka (format szesnastkowy): pierwsze cztery bajty to adres, kolejne osiem to dane.



Rysunek 13. Widok tworzonej aplikacji komputerowej w środowisku Microsoft Visual C#

Montaż

Schemat montażowy urządzenia pokazano na **rysunku 15**. Montaż warto rozpocząć od przytwierdzenia mikrokontrolera, co należy wykonać z należytą precyzją gwarantującą styczność wyprowadzeń z odpowiadającymi im padami płytki. Następną czynnością powinien być montaż pozostałych elementów SMD. W kolejnym kroku należy wlutować elementy przewlekane (gniazda, złącza, diody LED).

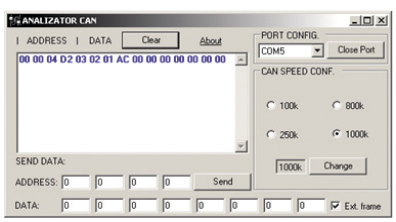
Opcjonalnie można dodatkowo osadzić urządzenie w obudowie (za pomocą przewidzianych otworów na śrubki) oraz przygotować przewód i złącze umożliwiające wygodne podłączanie analizatora do sieci CAN (**rysunek 16**).

Ostatnim etapem jest zaprogramowanie mikrokontrolera. Oprogramowanie zostało napisane przy wykorzystaniu środowiska Raisonance RIDE7 (kompatybilny z nim programator to R-Link), jednakże kod może zostać łatwo przeniesiony na inne platformy programistyczne.

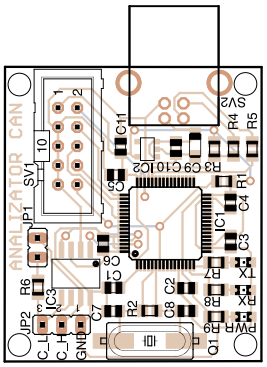
Podsumowanie

W artykule przedstawiono autorski projekt urządzenia diagnostycznego do sieci CAN. Urządzenie może być realną alternatywą dla istniejących rozwiązań komercyjnych, gdyż cechuje się zbliżoną do nich funkcjonalnością i niezawodnością. Dodatkowym atutem opracowanej konstrukcji jest prostota budowy, gdyż wykorzystano minimalną ilość elementów elektronicznych, które jednocześnie są tanie i łatwo dostępne na rynku.

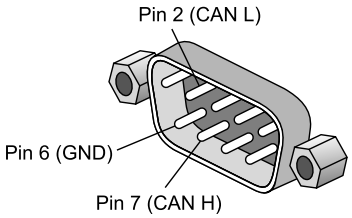
Istnieje możliwość dalszego ulepszania projektu. Interesującym kierunkiem rozwoju jest np. uzupełnienie aplikacji komputerowej o obsługę popularnych protokołów CAN warstwy 7. modelu OSI/ISO, które wymienione zostały na początku artykułu. Umożliwiłoby to obiektową interpretację danych przesyłanych przez sieć i wizualizację ich w formie konkretnych parametrów, ich wartości oraz jednostki. Istotną kwestią może być ponadto dodanie obsługi dodatkowych prędkości transmisji CAN, które w obecnej chwili są ograniczone do 4 wartości.



Rysunek 14. Widok aplikacji komputerowej w trakcie pracy



Rysunek 15. Schemat montażowy analizatora CAN



Rysunek 16. Wykonany na podstawie gniazda DB9 interfejs komunikacyjny CAN

Szymon Panecki
szymon.panecki@pwr.wroc.pl

Bibliografia:
Daniluk A. USB Praktyczne programowanie z Windows API w C++, 2009
Etschberger K. Controller Area Network: basics, protocols, chips and applications, 2001
Mielczarek W. USB Uniwersalny interfejs szeregowy, 2005
Voss W. A Comprehensible Guide to Controller Area Network, 2005
Yin J. The Definitive Guide to the ARM Cortex-M3, 2007
www.arm.com SWD – Serial Wire Debug
www.microchip.com MCP1802 datasheet
www.st.com STM32F10xxx hardware development: getting started
www.st.com STM32 datasheet
www.ti.com SN65HVD230 datasheet

Listing 2 Realizacja konwersji USB=>CAN

```
void EP3_OUT_Callback(void)
{
         USB_SIL_Read(EP3_OUT, USB_Rx_Buffer);    //odczyt USB

    if(USB_Rx_Buffer[12] == 1)                    //adres (rozszerzony)
    {
          TxMessage.ExtId = (USB_Rx_Buffer[0] << 24);
       TxMessage.ExtId = TxMessage.ExtId + (USB_Rx_Buffer[1] << 16);
       TxMessage.ExtId = TxMessage.ExtId + (USB_Rx_Buffer[2] << 8);
       TxMessage.ExtId = TxMessage.ExtId + USB_Rx_Buffer[3];
    }

    if(USB_Rx_Buffer[12] == 0)                //adres (podstawowy)
    {
       TxMessage.StdId = (USB_Rx_Buffer[0] << 24);
       TxMessage.StdId = TxMessage.ExtId + (USB_Rx_Buffer[1] << 16);
       TxMessage.StdId = TxMessage.ExtId + (USB_Rx_Buffer[2] << 8);
       TxMessage.StdId = TxMessage.ExtId + USB_Rx_Buffer[3];
    }

    TxMessage.Data[0]=USB_Rx_Buffer[4];       //dane
    TxMessage.Data[1]=USB_Rx_Buffer[5];
    TxMessage.Data[2]=USB_Rx_Buffer[6];
    TxMessage.Data[3]=USB_Rx_Buffer[7];
    TxMessage.Data[4]=USB_Rx_Buffer[8];
    TxMessage.Data[5]=USB_Rx_Buffer[9];
    TxMessage.Data[6]=USB_Rx_Buffer[10];
    TxMessage.Data[7]=USB_Rx_Buffer[11];
    TxMessage.IDE= USB_Rx_Buffer[12];         //rodzaj ramki
    TxMessage.RTR= USB_Rx_Buffer[13];
    TxMessage.DLC= USB_Rx_Buffer[14];         //dlugosc pola danych

    TransmitMailbox = CAN_Transmit(CAN1,&TxMessage);
    while(CAN_TransmitStatus(CAN1, TransmitMailbox) != CANTXOK)
    {
    }
    GPIO_SetBits(GPIOA, GPIO_Pin_2);
    my_delay(20000);
    GPIO_ResetBits(GPIOA, GPIO_Pin_2);
}
```