

# OK. Vama je potrebna ... CAN mreža



EasyAVR5A razvojni sistem i CAN-SPI moduli

Neretko se javlja potreba da se više mikrokontrolera koji vrše različite operacije integrišu u jedan sistem kako bi funkcionisali kao celina. U ovom tekstu ćemo pokazati kako umrežiti tri mikrokontrolera u CAN mrežu. Takođe, objasnićemo kako se koriste filtri u CAN čvorovima u cilju selekcije poruka.

Zoran Ristic  
MikroElektronika - Sektor za razvoj softvera

Obzirom da više perifernih jedinica koristi istu magistralu za razmenu podataka, neophodno je uvesti red u način korišćenja te magistrale. CAN standard precizno opisuje sve detalje umrežavanja više uređaja i kao takav je široko prihvaćen u industriji. CAN standard precizno definiše prvenstvo korišćenja magistrale i hardverski rešava problem „sudara“ u slučaju da više perifernih jedinica počne da komunicira u isto vreme.

## Hardver

U našem primeru CAN mreža će biti konfigurisana tako da prvi uređaj emituje poruke sa ID-jem 0x10 i 0x11, a drugi i treći će emitovati poruke sa ID-jem 0x12 i 0x13 respektivno. Takođe, CAN čvorovi će biti konfigurisani tako da drugi čvor odgovara samo na dolazne pakete sa ID-jem 0x10, a treći uređaj odgovara samo na one pakete čiji je ID jednak 0x11. Konačno, prvi uređaj je podešen tako da prima poruke sa ID-jem 0x12 i 0x13 (slika 2.). Filtriranje poruka ćemo lako podesiti tako što ćemo pozvati rutinu CANSPISetFilter koja će izvršiti sva neophodna podešavanja registra mikrokontrolera i CAN SPI pločice.

Generalno, CAN standard ne zahteva prisustvo master uređaja na magistrali,

ali ćemo mi u cilju lakšeg razumevanja primera i bez gubitka opštosti podesiti da samo prvi uređaj inicira komunikaciju na mreži, a da preostala dva uređaja odgovaraju na pojedinačne prozivke.

## Softver

Prilikom emisije poruke master čvor ostavlja dovoljno vremena da se prozvani čvor odazove. U slučaju da nakon isteka tog vremena udaljeni čvor ne pošalje odgovor, tada master proglašava grešku za tekuću poruku i nastavlja dalje sa prozivkom ostalih čvorova (slika 3.). U slučaju da nakon isteka očekivanog vremena periferni CAN čvor ipak krene u odgovor u isto vreme kada i neki drugi čvor, tada će doći do kolizije na CAN magistrali. Međutim, prioritet adresa uređaja i koncepcija CAN mreže je takva da će se uređaj nižeg prioriteta u ovom slučaju povući sa magistrale čime infor-

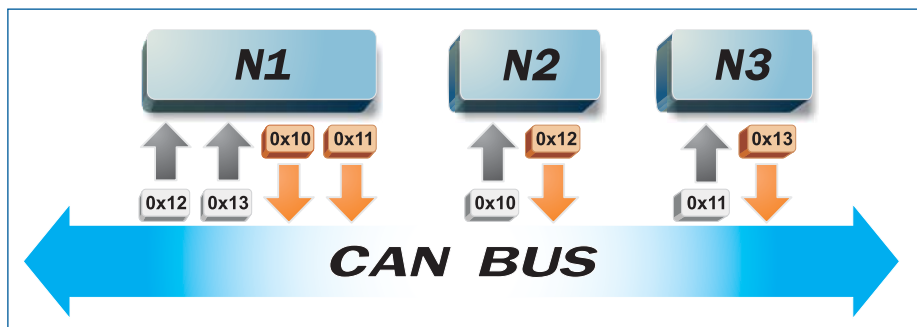
macija višeg prioriteta neometano nastavlja da se emituje preko mreže.

Kao što smo napomenuli, iskoristićemo interni SPI modul mikrokontrolera za slanje podataka na CAN magistralu. Neke od prednosti korišćenja internog SPI modula mikrokontrolera su:

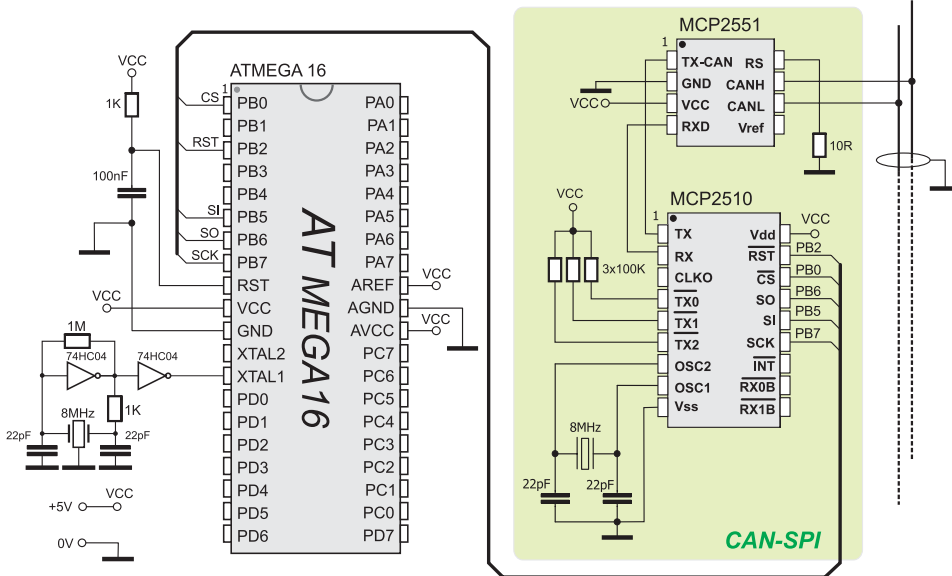
1. Mogućnost generisanja prekida pri likom predaje i prijema podataka;
2. Nezavisnost rada SPI modula od ostalih periferija mikrokontrolera; i
3. Jednostavna konfiguracija.

CAN SPI biblioteka omogućava podešavanje režima rada CAN mreže, postavljanje filtera za predajne i prijemne čvorove, čitanje podataka iz bafera CAN SPI pločice itd.

Priloženi primer uključuje LED na pinovima mikrokontrolera i time daje in-



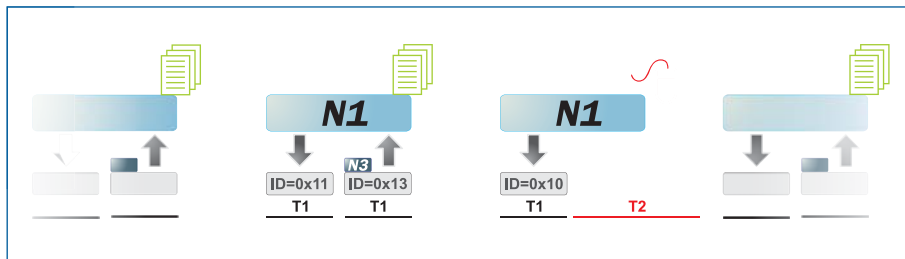
Slika 1. Filtriranje poruka



Šema 1. Povezivanje CAN-SPI modula sa ATmega 16 mikrokontrolerom

dikaciju da je mreža funkcionalna. Kada se čvor broj 2 odazove na poziv čvora broj 1, uključuju se LED diode na portu B. U slučaju da čvor broj 3 odgovori na poziv, uključuju se LED diode na portu D.

U primeru je dat izvorni kod za sva tri čvora u mreži. Da bi se napravio HEX za svaki od čvorova pojedinačno, potrebno je ostaviti uvek jednu i samo jednu DEFINE direktivu u zaglavlju primera.



Slika 2. Primer komunikacije

U ovom primeru smo predstavili jedan način umrežavanja mikrokontrolera u CAN mrežu. Na primeru komunikacionog protokola smo objasnili kako se vrši detekcija grešaka u slučaju da udaljeni čvor ne šalje očekivane informacije. Takođe, pokazali smo kako se filtriraju poruke pomoću CAN filtera i kako se generalno vrši komunikacija na CAN magistrali.

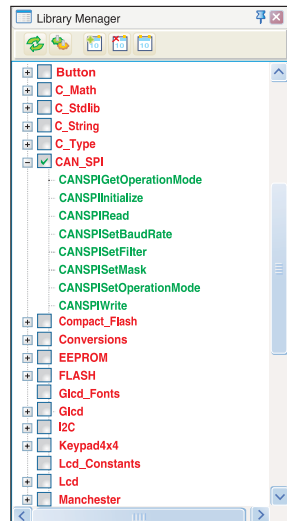
Editor biblioteka kompajlera mikroPASCAL PRO for AVR® sa gotovim bibliotekama kao što su: CAN\_SPI, GLCD, Ethernet itd.

### Funkcije koje se koriste u programu

- CANSPIGetOperationMode() Aktivni mod rada
- CANSPIInitialize()\* Inicijalizacija CANSPI modula
- CANIRead()\* Čitanje poruke
- CANSPISetBaudRate() Podešavanje CANSPI baud rejta
- CANSPISetFilter()\* Konfiguracija filtera za poruke
- CANSPISetMask()\* Napredna konfiguracija filtera
- CANSPISetOperationMode()\* Aktivni mod rada
- CANSPIWrite()\* Upis poruke

#### \* Funkcije CANSPI biblioteke korišćene u programu

- Ostale funkcije kompajlera mikroPASCAL PRO for AVR korišćene u programu:
- Delay\_us()
- SPI1\_init()
- SPI1\_read()



### Program za demonstriranje rada CAN magistrale

```

program CanSpi;
{ Description: This program demonstrates how to make a CAN
network using mikroElektronika
CANSPI boards and mikroPascal compiler.
Target device: Microchip ATMEGA AVR ATMEGA 16
Oscillator: 8MHz crystal }

var Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags : byte;
    // can flags
    Rx_Data_Len : byte;
    // received data length in bytes
    RxDx_Data : array[8] of byte;
    // can rx/tx data buffer
    Msg_Rcvd : byte;
    // reception flag
    Tx_ID, Rx_ID : longint;

    // can rx and tx ID
    ErrorCount : byte;
// CANSPI module connections
var CanSpi_CS : sbit at PORTB.B0;
    // chip select (CS) pin for CANSPI board
    CanSpi_CS_Direction : sbit at DDRB.B0;
    // Direction register for CS pin
    CanSpi_Rst : sbit at PORTB.B2;
    // Reset pin for CANSPI board
    CanSpi_Rst_Direction : sbit at DDRB.B2;
    // Direction register for Reset pin
// End CANSPI module connections
begin
    ADCSRA.7 := 0;
    // Configure analog pins as digital I/O
    PORTB := 0; DDRB := 0;
    // Initialize ports
    PORTC := 0; DDRD := 0;
    PORTC := 0; DDRC := 255;
    ErrorCount := 0;
    // Error flag
    Can_Init_Flags := 0; Can_Send_Flags := 0; Can_Rcv_Flags := 0;
    // clear flags

    Can_Send_Flags := _CANSPI_TX_PRIORITY_0 and
    // form value to be used
    _CANSPI_TX_XTD_FRAME and
    // with CANSPIwrite
    _CANSPI_TX_NO_RTR_FRAME;

    Can_Init_Flags := _CANSPI_CONFIG_SAMPLE_THRICE and
    // form value to be used
    _CANSPI_CONFIG_FHSEG2_PRG_ON and
    // with CANSPIInit
    _CANSPI_CONFIG_XTD_MSG and
    _CANSPI_CONFIG_DBG_BUFFER_ON and
    _CANSPI_CONFIG_VALID_XTD_MSG;

    SPI1_Init();
    // initialize SPI module
    CANSPIInitialize(1, 3, 3, 1, Can_Init_Flags);
    // Initialize external CANSPI module
    CANSPISetOperationMode(_CANSPI_MODE_CONFIG, TRUE);
    // set CONFIGURATION mode
    CANSPISetMask(_CANSPI_MASK_B1, -1, _CANSPI_CONFIG_XTD_MSG);
    // set all mask1 bits to ones
    CANSPISetMask(_CANSPI_MASK_B2, -1, _CANSPI_CONFIG_XTD_MSG);
    // set all mask2 bits to ones

    CANSPISetFilter(_CANSPI_FILTER_B2_F4, 0x12, _CANSPI_CONFIG_XTD_MSG);
    // Node1 accepts messages with ID 0x12
    CANSPISetFilter(_CANSPI_FILTER_B1_F1, 0x13, _CANSPI_CONFIG_XTD_MSG);
    // Node1 accepts messages with ID 0x13

    CANSPISetOperationMode(_CANSPI_MODE_NORMAL, 0xFF);
    // set NORMAL mode
    RxDx_Data[0] := 0x40;
    // set initial data to be sent

    Tx_ID := 0x10;
    // set transmit ID for CAN message

    CANSPIWrite(Tx_ID, RxDx_Data, 1, Can_Send_Flags);
    // Node1 sends initial message

    while (TRUE) do
    // endless loop
    begin
        Msg_Rcvd := CANSPIRead(Rx_ID, RxDx_Data, Rx_Data_Len,
        Can_Rcv_Flags); // attempt receive message
        if (Msg_Rcvd) then begin
            // if message is received then check id

            if Rx_ID = 0x12 then
            // check ID
            PORTC := RxDx_Data[0]
            // output data at PORTC
            else
            PORTD := RxDx_Data[0];
            // output data at PORTD
            delay_ms(50);
            // wait for a while between messages
            CANSPIWrite(Tx_ID, RxDx_Data, 1, Can_Send_Flags);

            // send one byte of data
            inc(Tx_ID);
            // switch to next message
            if Tx_ID > 0x11 then Tx_ID := 0x10;
            // check overflow

            end
        else begin
            // an error occurred, wait for a while

            inc(ErrorCount);
            // increment error indicator
            Delay_ms(10);
            // wait for 10ms
            if (ErrorCount > 10) then begin
                // timeout expired - process errors
                ErrorCount := 0;
                // reset error counter
                inc(Tx_ID);
                // switch to another message
                if Tx_ID > 0x11 then Tx_ID := 0x10;
                // check overflow
                CANSPIWrite(Tx_ID, RxDx_Data, 1, Can_Send_Flags);
                // send new message
            end;
        end;
    end;
end.
    
```



GO TO Ovaj program, pisan za AVR® mikrokontrolere u programima C, Basic i Pascal kao i programe napisane za mikrokontrolere PIC®, dsPIC i 8051® možete naći na našem web sajtu: [www.mikroe.com/en/article/](http://www.mikroe.com/en/article/)