

OK. Vama je potrebna ... CAN mreža



Easypic4A 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-

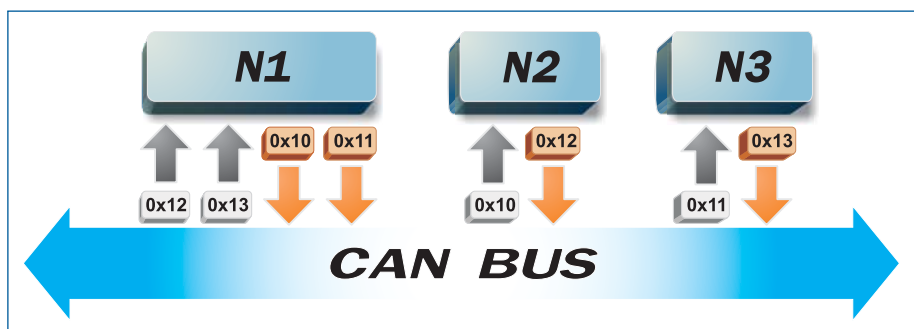
macija višeg prioriteta neometano nastavlja da se emituje preko mreže.

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

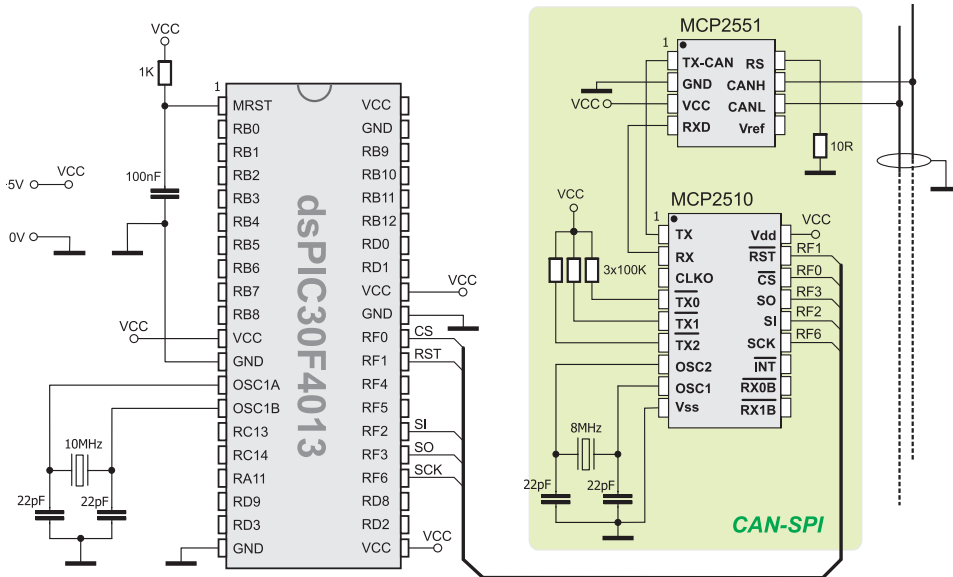
1. Mogućnost generisanja prekida pri likom predaje i prijema podataka;
2. Nezavisnost rada SPI modula od ostalih periferija mikrokontrolera; i
3. Jednostavna konfiguracija.

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 dsPIC30F4013 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.

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 for
dsPIC.
Target device: Microchip dsPIC 30F4013
Oscillator: 10MHz crystal }

var Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags : byte;
    // can flags
Rx_Data_Len : byte;
    // received data length in bytes
RxTx_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;

begin
ADPCFG := 0xFFFF;
    // Configure analog pins as digital I/O
PORTB := 0; TRISB := 0;
    // Initialize ports
PORTD := 0; TRISD := 0;
TRISF := 0;
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_PHSSEG2_PRG_ON and
    // with CANSPIInit
CANSPI_CONFIG_XTD_MSG and
CANSPI_CONFIG_DBL_BUFFER_ON and
CANSPI_CONFIG_VALID_XTD_MSG;

SPI1_Init();

CANSPI1Init(1,3,3,1,Can_Init_Flags, PORTF, 1, PORTF, 0);
    // Initialize external CANSPI module

CANSPI1SetOperationMode(CANSPI_MODE_CONFIG, TRUE);
    // set CONFIGURATION mode
CANSPI1SetMask(CANSPI_MASK_B1, -1, CANSPI_CONFIG_XTD_MSG);
    // set all mask1 bits to ones
CANSPI1SetMask(CANSPI_MASK_B2, -1, CANSPI_CONFIG_XTD_MSG);
    // set all mask2 bits to ones

CANSPI1SetFilter(CANSPI_FILTER_B2_F4, 0x12, CANSPI_CONFIG_XTD_MSG);
    // Node1 accepts messages with ID 0x12
CANSPI1SetFilter(CANSPI_FILTER_B1_F1, 0x13, CANSPI_CONFIG_XTD_MSG);
    // Node1 accepts messages with ID 0x13

CANSPI1SetOperationMode(CANSPI_MODE_NORMAL, 0xFF);
    // set NORMAL mode
RxTx_Data[0] := 0x40;
    // set initial data to be sent

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

CANSPI1Write(Tx_ID, RxTx_Data, 1, Can_Send_Flags);
    // Node1 sends initial message

while (TRUE) do
    // endless loop
begin
Msg_Rcvd := CANSPI1Read(Rx_ID, RxTx_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
PORTB := RxTx_Data[0]
    // output data at PORTB
else
PORTD := RxTx_Data[0];
    // output data at PORTD
delay_ms(50);
    // wait for a while between messages
CANSPI1Write(Tx_ID, RxTx_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

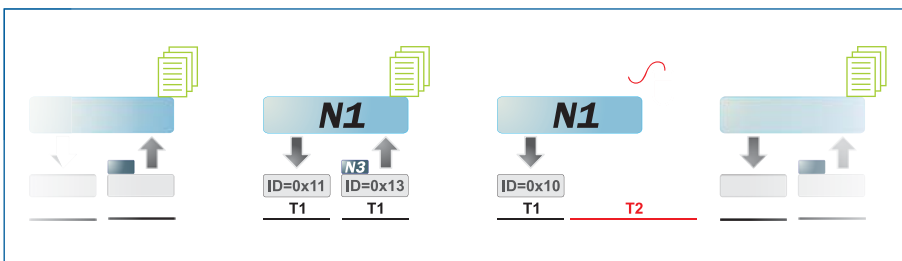
end
else begin
    // an error occured, wait for a while

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
CANSPI1Write(Tx_ID, RxTx_Data, 1, Can_Send_Flags);
    // send new message

end;

end;

end;
end.
    
```

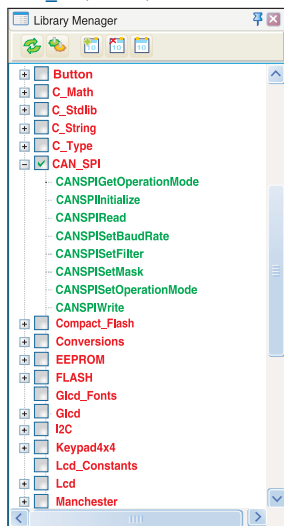


Sluka 2. Primer komunikacije

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 for dsPIC® 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 for dsPIC korišćene u programu:

- Delay_us()
- SPI1_init()
- SPI1_read()

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

