

OK. Vama je potrebna ... CAN mreža



Tri EasyPIC5 razvojna sistema povezana na CAN magistralu preko CAN-SPI modula

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 podešiti 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-

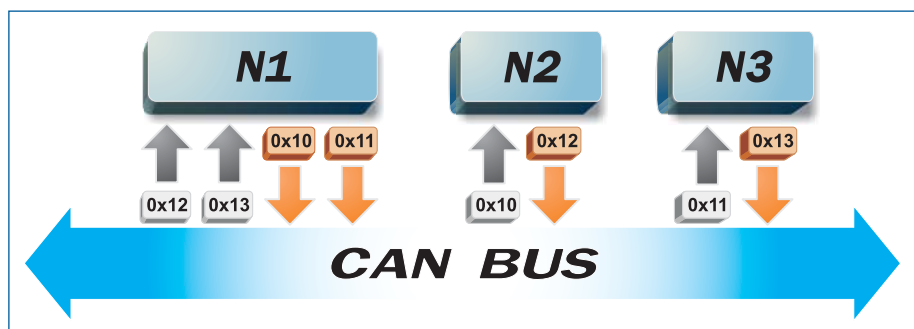
macija višeg prioriteta neometano nastavlja da se emituje preko mreže.

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

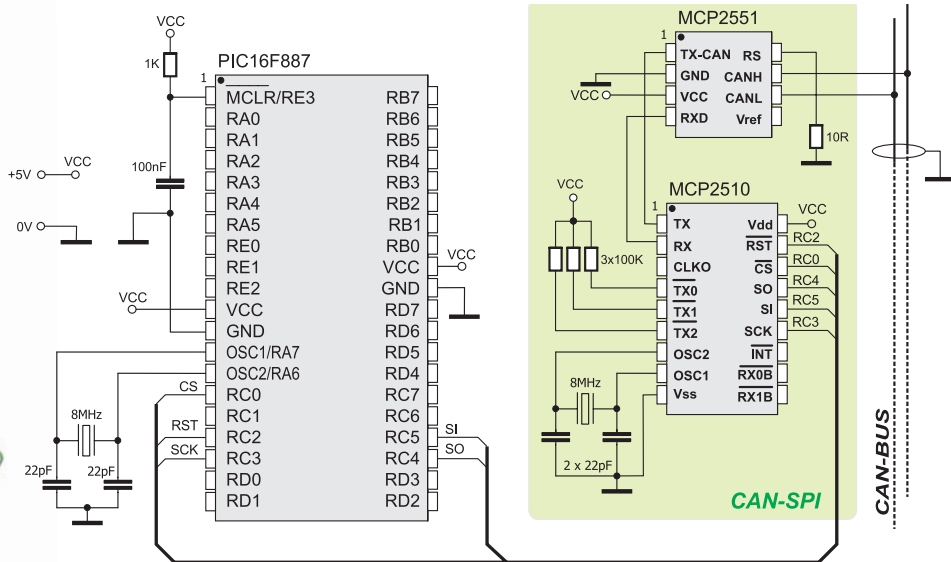
1. Mogućnost generisanja prekida pri likom predaje i prijema podataka;
2. Nezavisnost rada SPI modula od ostalih perifera mikrokontrolera; i
3. Jednostavna konfiguracija.

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-



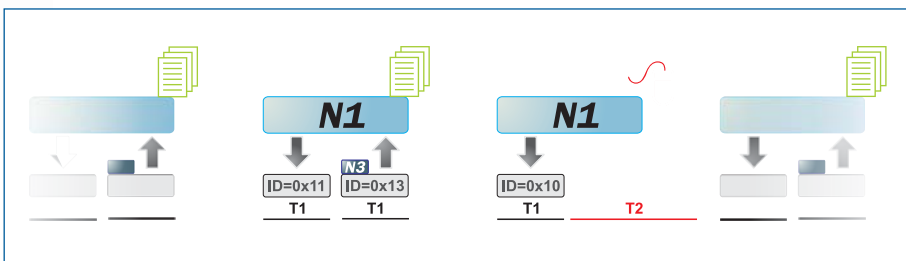
Slika 1. Filtriranje poruka



Šema 1. Povezivanje CAN-SPI modula sa PIC16F887 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.

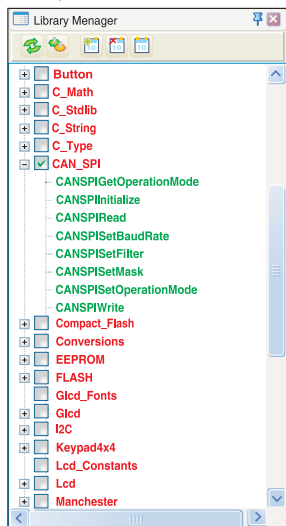


Slika 2. Primer komunikacije

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 mikroC PRO for PIC® 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 mikroC PRO for PIC korišćene u programu:

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

Program za demonstriranje rada CAN magistrale

```
#define NODE1 // Uncomment this line to build HEX for Node 1
// #define NODE2 // Uncomment this line to build HEX for Node 2
// #define NODE3 // Uncomment this line to build HEX for Node 3
char Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags; // Can flags
char RxDx_Data_Len; // Received data length in bytes
char RxDx_Data[8]; // Can rx/tx data buffer
char Msg_Rcvd; // Reception flag
long Tx_ID, Rx_ID; // Can rx and tx ID
char ErrorCount;
// CANSPI module connections
sbit CanSpi_CS at RC0_bit; // Chip select (CS) pin for CANSPI board
sbit CanSpi_CS_Direction at TRISC0_bit; // Direction register for CS pin
sbit CanSpi_Rst at RC2_bit; // Reset pin for CANSPI board
sbit CanSpi_Rst_Direction at TRISC2_bit; // Direction register for Reset pin
// End CANSPI module connections
void main() {
  ANSEL = 0; ANSELH = 0; // Configure analog pins as digital I/O
  PORTB = 0; TRISB = 0; // Initialize ports
  PORTD = 0; TRISD = 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 & // Form value to be used
  _CANSPI_TX_XID_FRAME & // with CANSPIwrite
  _CANSPI_TX_NO_RTR_FRAME;

  Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE & // Form value to be used
  _CANSPI_CONFIG_PHSIG2_PRG_ON & // with CANSPIInit
  _CANSPI_CONFIG_XID_MSG &
  _CANSPI_CONFIG_DBL_BUFFER_ON &
  _CANSPI_CONFIG_VALID_XID_MSG;

  SPI_Init(); // Initialize SPI module
  CANSPIInitialize(1, 3, 3, 1, Can_Init_Flags);
  // Initialize external CANSPI module
  CANSPISetOperationMode(_CANSPI_MODE_CONFIG, 0xFF);
  // Set CONFIGURATION mode
  CANSPISetMask(_CANSPI_MASK_B1, -1, _CANSPI_CONFIG_XID_MSG);
  // Set all mask1 bits to ones
  CANSPISetMask(_CANSPI_MASK_B2, -1, _CANSPI_CONFIG_XID_MSG);
  // Set all mask2 bits to ones
  #ifdef NODE1
  CANSPISetFilter(_CANSPI_FILTER_B2_F4, 0x12, _CANSPI_CONFIG_XID_MSG);
  // Node1 accepts messages with ID 0x12
  CANSPISetFilter(_CANSPI_FILTER_B1_F1, 0x13, _CANSPI_CONFIG_XID_MSG);
  // Node1 accepts messages with ID 0x13
  #else
  CANSPISetFilter(_CANSPI_FILTER_B2_F2, 0x10, _CANSPI_CONFIG_XID_MSG);
  // Node2 and Node3 accept messages with ID 0x10
  CANSPISetFilter(_CANSPI_FILTER_B1_F2, 0x11, _CANSPI_CONFIG_XID_MSG);
  // Node2 and Node3 accept messages with ID 0x11
  #endif
  CANSPISetOperationMode(_CANSPI_MODE_NORMAL, 0xFF); // Set NORMAL mode
  RxDx_Data[0] = 0x40; // Set initial data to be sent
  #ifdef NODE1
  Tx_ID = 0x10; // Set transmit ID for CAN message
  #endif
  #ifdef NODE2
  Tx_ID = 0x12; // set transmit ID for CAN message
  #endif
  #ifdef NODE3
  Tx_ID = 0x13; // Set transmit ID for CAN message
  #endif
  #ifdef NODE1
  CANSPIWrite(Tx_ID, &RxDx_Data, 1, Can_Send_Flags);
  // Node1 sends initial message
  #endif
  while (1) // Endless loop
  {
    Msg_Rcvd = CANSPIRead(&Rx_ID, &RxDx_Data, &RxDx_Data_Len, &Can_Rcv_Flags);
    // Attempt receive message
    if (Msg_Rcvd) { // If message is received then check id
      #ifdef NODE1
      if (Rx_ID == 0x12) // Check ID
        PORTB = RxDx_Data[0]; // Output data at PORTB
      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
      Tx_ID++; // Switch to next message
      if (Tx_ID > 0x11) Tx_ID = 0x10; // Check overflow
      #endif
      #ifdef NODE2
      if (Rx_ID == 0x10) { // Check if this is our message
        PORTB = RxDx_Data[0]; // Display incoming data on PORTB
        RxDx_Data[0] = RxDx_Data[0] << 1; // Prepare data for sending back
        if (RxDx_Data[0] == 0) RxDx_Data[0] = 1; // Reinitialize if
        //maximum reached
      }
      delay_ms(10); // Wait for a while
      CANSPIWrite(Tx_ID, &RxDx_Data, 1, Can_Send_Flags); // Send one byte
      //of data back
      }
      #endif
      #ifdef NODE3
      if (Rx_ID == 0x11) { // Check if this is our message
        PORTD = RxDx_Data[0]; // Display incoming data on PORTD
        RxDx_Data[0] = RxDx_Data[0] >> 1; // Prepare data for sending back
        if (RxDx_Data[0] == 0) RxDx_Data[0] = 128; // Reinitialize if
        //maximum reached
      }
      delay_ms(10); // Wait for a while
      CANSPIWrite(Tx_ID, &RxDx_Data, 1, Can_Send_Flags); // Send one byte
      //of data back
      }
      #endif
    }
  }
  #ifdef NODE1
  #endif
  #ifdef NODE2
  #endif
  #ifdef NODE3
  #endif
  ErrorCount++; // An error occurred, wait for a while
  ErrorCount++; // Increment error indicator
  delay_ms(10); // Wait for 100ms
  if (ErrorCount > 10) { // Timeout expired - process errors
    ErrorCount = 0; // Reset error counter
    Tx_ID++; // Switch to another message
    if (Tx_ID > 0x11) Tx_ID = 0x10; // Check overflow
    CANSPIWrite(Tx_ID, &RxDx_Data, 1, Can_Send_Flags);
    // Send new message
  }
  #ifdef NODE1
  #endif
}
```

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

