

# Muy bien. Lo que necesitamos ahora es un ... bus CAN



Tres sistemas de desarrollo EasyPIC5 conectados a un bus CAN por medio de módulos CAN-SPI.

A menudo es necesario disponer de varios microcontroladores realizando diferentes operaciones, integrados en un sistema, haciendo que todos funcionen como uno solo. En este caso vamos a mostrar cómo conectar tres microcontroladores a un bus CAN y cómo usar los filtros en los nodos CAN para conseguir hacer un filtrado de mensajes.

Por Zoran Ristic

Departamento de Software – MikroElektronika

En los casos en los que varias unidades periféricas comparten el mismo bus de datos, es necesario definir cómo acceder a dicho bus. El protocolo CAN describe, de manera precisa, todos los detalles de cómo conectar varios dispositivos a una red y ésta es la forma en la que se usa ampliamente en la industria. El protocolo define principalmente la prioridad de la implementación del bus y soluciona el problema de "colisión" dentro del circuito, en el caso de que varios periféricos inicien su comunicación al mismo tiempo.

## El Circuito

En este ejemplo, un bus CAN será configurado de manera que el primer dispositivo envíe mensajes que consisten de 0x10 y 0x11, como su ID, mientras que el segundo y el tercer dispositivo envían mensajes que consisten de los IDs 0x12 y 0x13, respectivamente. También vamos a configurar los nodos CAN de manera que el segundo nodo responda a los mensajes entrantes que contengan solo el ID 0x10, mientras que el tercero sólo responderá a aquellos mensajes que contengan el ID 0x11. Por consiguiente, el primer dispositivo está configurado para recibir mensajes que contengan un ID 0x12 y 0x13 (ver Figura 2). El filtrado de mensajes

se implementa fácilmente llamando a la rutina CANSPISetFilter, la cual también se encargará de manejar todas las configuraciones necesarias de los registros del microcontrolador y de la placa CAN SPI.

En general, el protocolo CAN no requiere que haya un dispositivo Maestro presente en el bus. Sin embargo, para conseguir que este ejemplo sea más fácil de entender, al mismo tiempo que nos sirve de propósito general, sólo vamos a configurar el primer dispositivo para iniciar la comunicación en la red y los otros dos dispositivos para responder a las llamadas individuales.

## El Programa

Cuando se envía un mensaje, el nodo Maestro deja transcurrir suficiente tiempo para que el nodo llamado responda. En el caso en que un nodo re-

moto no responda dentro del tiempo requerido, el dispositivo Maestro informa de un error en el mensaje actual y procede con la llamada a otros nodos (ver Figura 3). En el caso de que un nodo CAN periférico responda al mismo tiempo que otro nodo, se producirá una "colisión" en el bus CAN. Sin embargo, la dirección prioritaria del dispositivo y el propio bus CAN establecen que, en este caso, el nodo que transmite el mensaje con la prioridad más baja lo retira del bus, lo que permite que el nodo que transmite el mensaje con la prioridad más alta proceda con su transmisión de manera inmediata. Como hemos mencionado anteriormente, usaremos un módulo SPI interno del microcontrolador para transferir datos sobre el bus CAN. Algunas de las ventajas de usar el módulo SPI interno del microcontrolador son: la posibilidad de generar

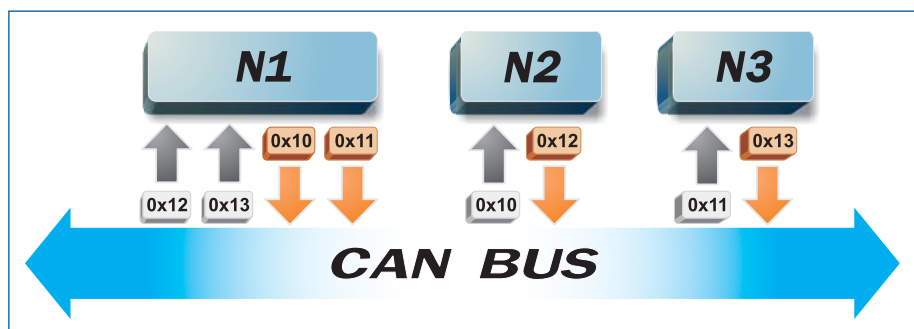
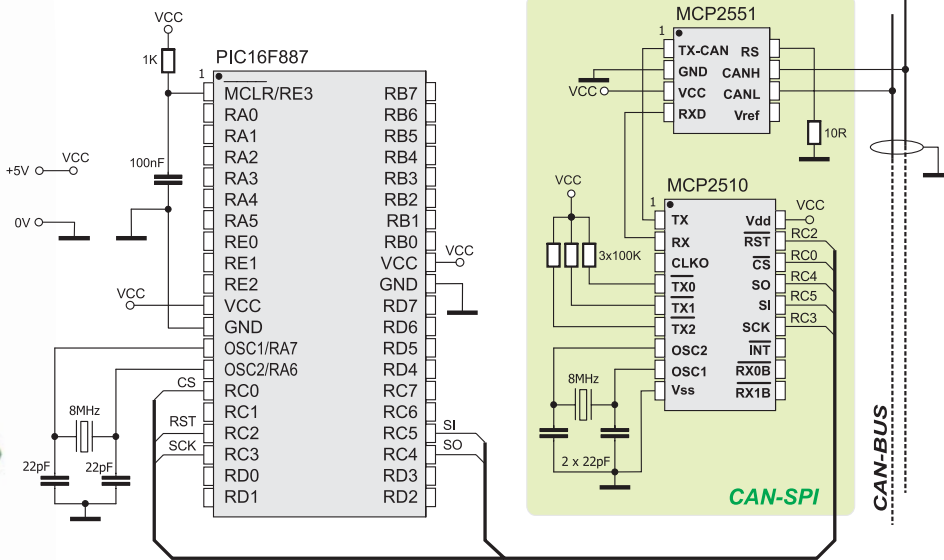


Figura 1. Filtrado de mensajes



Programa para demostrar el funcionamiento de una red CAN

```
#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() {
  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;

  SPI1_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 == 0x11) { // 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
    }
    else { // An error occurred, wait for a while
      #ifdef NODE1
      ErrorCount++; // Increment error indicator
      Delay_ms(10); // Wait for 10ms
      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
      }
      #endif
    }
  }
}
```

Esquema eléctrico 1. Conexión del módulo CAN-SPI al PIC16F887

una interrupción cuando se envía o se recibe datos; el módulo SPI funciona independientemente de otros periféricos y dispone de una configuración simple. La librería CAN SPI nos permite configurar el modo de funcionamiento del bus CAN y los filtros del nodo, leer datos desde el "buffer" de la placa CAN SPI, etc. Este ejemplo también incluye diodos LED en los terminales del microcontrolador que indican que la red funciona adecuadamente. Cuando el nodo 2 responde a la llamada del nodo 1, los LED del PORTB se encenderán. El código fuente para los tres nodos de la red se proporciona con este ejemplo. Para poder crear un fichero HEX para cada uno de estos nodos, de manera individual, es necesario escribir tan sólo una directiva DEFINE en la cabecera del ejemplo.

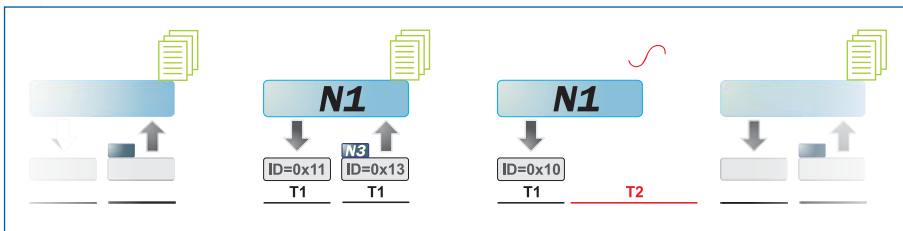
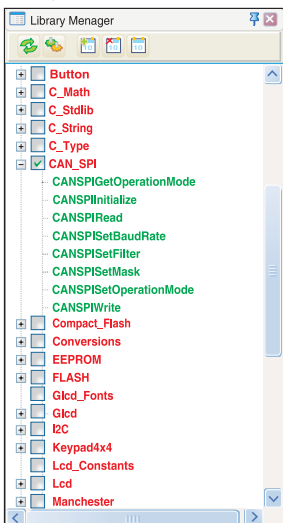


Figura 2. Ejemplo de comunicación

En resumen, hemos descrito una manera de conectar microcontroladores al bus CAN. También hemos descrito como detectar errores por medio de un protocolo de comunicación, en el caso en que un nodo remoto no responda como se esperaba, como filtrar mensajes usando los filtros CAN, además de cómo establecer comunicación, de manera general, sobre el bus CAN.

Editor de librerías mikroC PRO para PIC® con librerías, listas para ser usadas, como: CAN\_SPI, GLCD, Ethernet etc.

### Funciones usadas en el programa



CANSPIGetOperationMode()	Modo de operación actual
CANSPIInitialize()*	Inicializa el módulo CANSPI
CANIRead()*	Lee el mensaje
CANSPISetBaudRate()	Establece la velocidad del CANSPI
CANSPISetFilter()*	Configura el filtro de mensajes
CANSPISetMask()*	Configuración de filtrado avanzada
CANSPISetOperationMode()*	Modo de operación actual
CANSPIWrite()*	Escribe el mensaje
<b>* Funciones de la librería CANSPI usadas en el programa</b>	
Otras funciones mikroC PRO para PICs usadas en el programa:	
Delay_us()	
SPI1_init()	
SPI1_read()	

**GO TO** El código para este ejemplo escrito para microcontroladores PIC® en C, Basic y Pascal, así como los programas escritos para microcontroladores dsPIC®, 8051 y AVR® los pueden encontrar en nuestra página web: [www.mikroe.com/en/article/](http://www.mikroe.com/en/article/).

