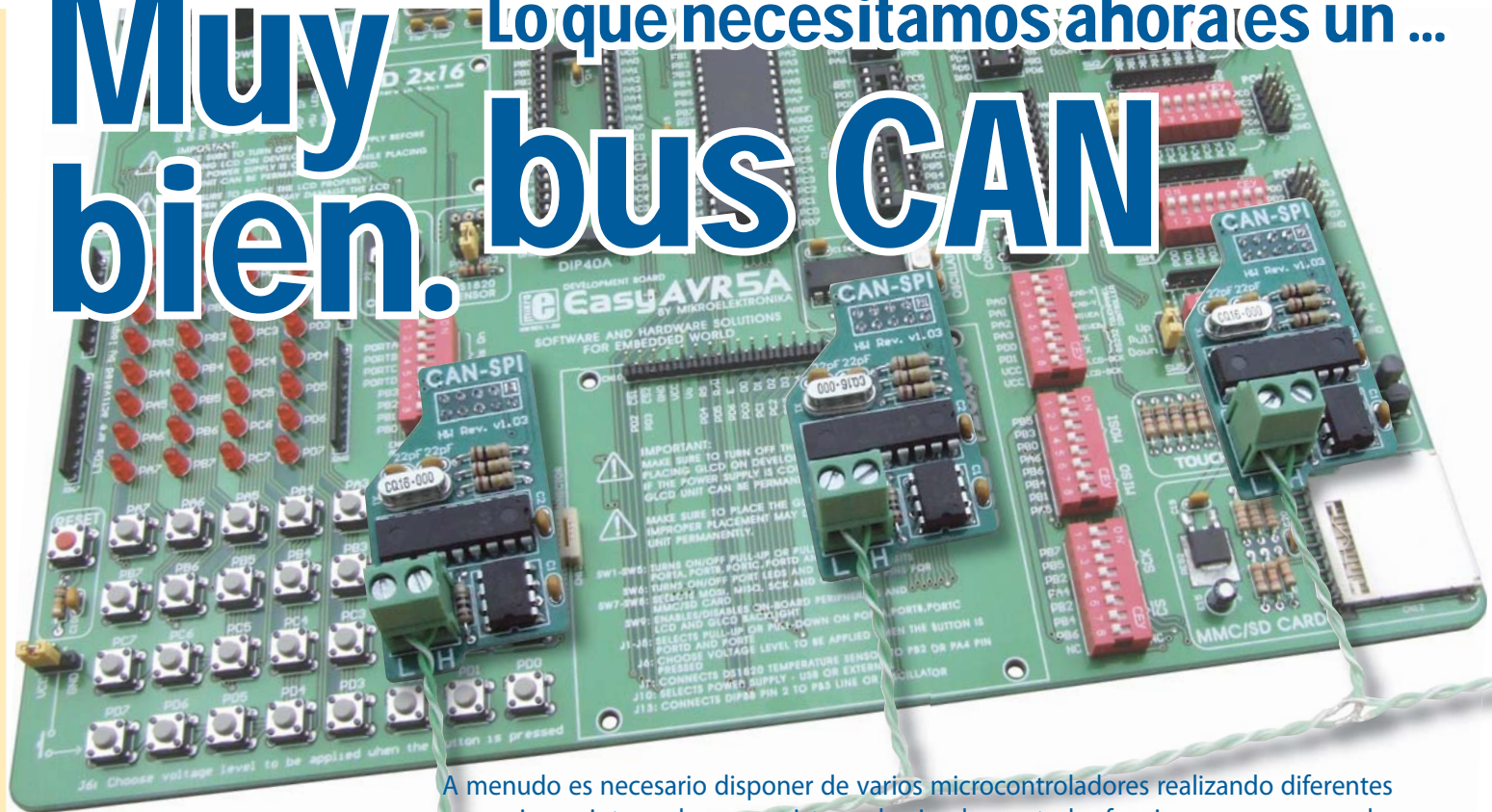


Muy bien. Lo que necesitamos ahora es un ... bus CAN



Sistemas de desarrollo EasyAVR5A y 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 remoto 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 una interrupción cuando se envía o se recibe datos; el módulo SPI funciona independientemente de otros periféricos y dispone de

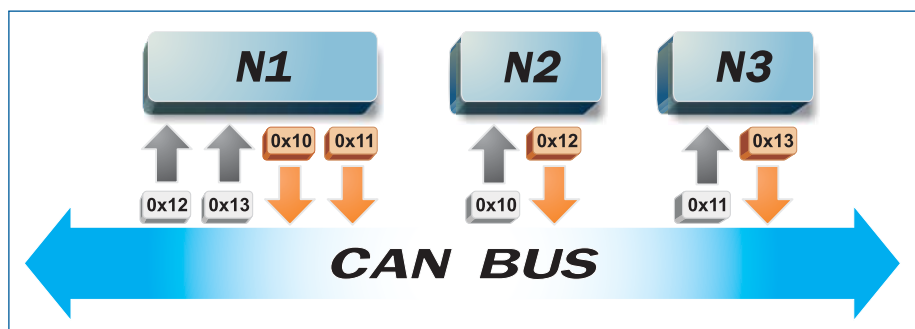
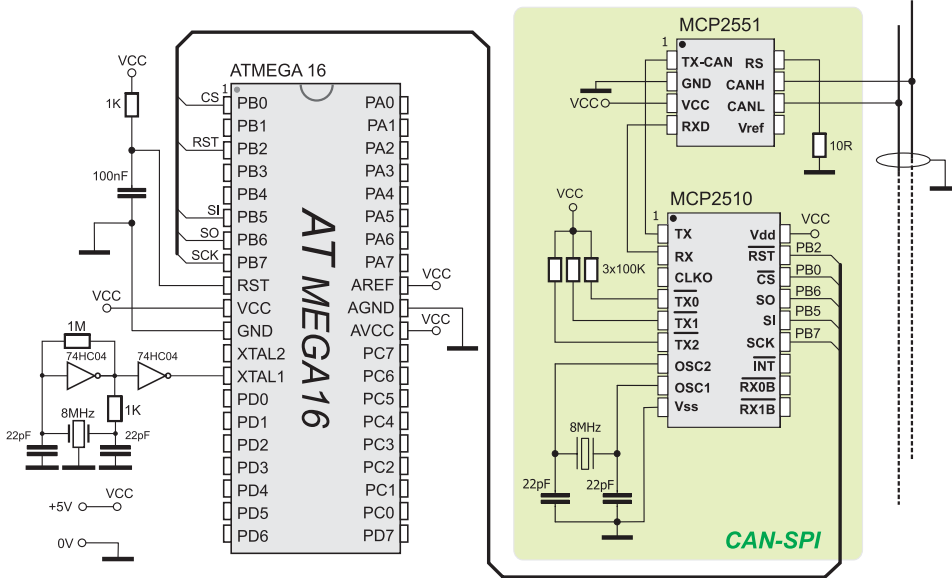


Figura 1. Filtrado de mensajes



Programa para demostrar el funcionamiento de una red CAN

```

program CanSPI
  ' Description: This program demonstrates how to make a CAN
  ' network using mikroElektronika
  ' * CANSPI boards and mikroBasic compiler.
  ' Target device: ATMEGA 16
  ' Oscillator: 8MHz crystal

  dim Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags as byte
  ' can flags
  Rx_Data_Len as byte
  ' received data length in bytes
  RxDx_Data as byte[8]
  ' can rx/tx data buffer
  Msg_Rcvd as byte
  ' reception flag
  Tx_ID, Rx_ID as longint
  ' can rx and tx ID
  ErrorCount as byte
  ' CANSPI module connections

  dim CanSpi_CS as sbit at PORTB.B0
  ' Chip select (CS) pin for CANSPI board
  CanSpi_CS_Direction as sbit at DDRB.B0
  ' Direction register for CS pin
  CanSpi_Rst as sbit at PORTB.B2
  ' Reset pin for CANSPI board
  CanSpi_Rst_Direction as sbit at DDRB.B2
  ' Direction register for Reset pin
  ' End CANSPI module connections
main:
  ADCSRA.7 = 0
  ' Configure analog pins as digital I/O
  PORTB = 0
  DDRB = 255
  ' Initialize ports
  PORTD = 0
  DDRD = 255
  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_PHSSEG2_PRG_ON and
  ' with CANSPIInit
  _CANSPI_CONFIG_XTD_MSG and
  _CANSPI_CONFIG_DBL_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)
  ' Model accepts messages with ID 0x12
  CANSPISetFilter(_CANSPI_FILTER_B1_F1, 0x13, _CANSPI_CONFIG_XTD_
  MSG)
  ' Model accepts messages with ID 0x13

  CANSPISetOperationMode(_CANSPI_MODE_NORMAL, 0x6F)
  ' set NORMAL mode
  RxDx_Data[0] = 0x40
  ' set initial data to be sent

  Tx_ID = 0x10
  ' set transmit ID for CAN message
  ' Model sends initial message

  CANSPIWrite(Tx_ID, RxDx_Data, 1, Can_Send_Flags)

  while (TRUE)
    ' endless loop
    Msg_Rcvd = CANSPIRead(Rx_ID, RxDx_Data, Rx_Data_Len,
    Can_Rcv_Flags) ' attempt receive message
    if (Msg_Rcvd) then
      ' 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
          end if
          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 end if
          ' check overflow
        else
          ' an error occurred, wait for a while
          inc(ErrorCount)
          ' increment error indicator
          Delay_ms(10)
          ' wait for 10ms
          if (ErrorCount > 10) then
            ' timeout expired - process errors
            ErrorCount = 0
            ' reset error counter
            inc(Tx_ID)
            ' switch to another message
            if Tx_ID > 0x11 then Tx_ID = 0x10 end if
            ' check overflow
            CANSPIWrite(Tx_ID, RxDx_Data, 1, Can_Send_Flags)
            ' send new message
            end if
          end if
        end if
      end
    end
  end.
  
```

Esquema eléctrico 1. Conexión del módulo CAN-SPI al ATmega 16

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.

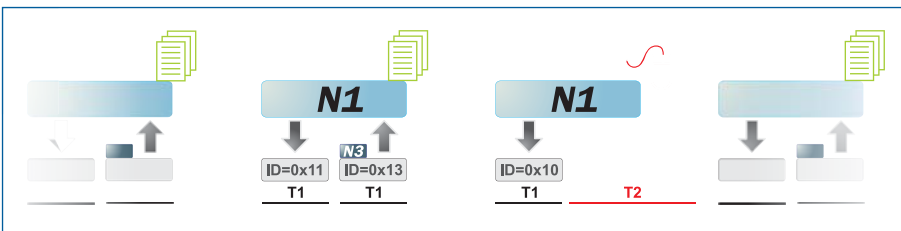
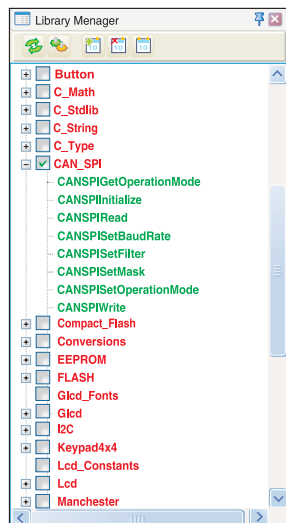


Figura 2. Ejemplo de comunicación

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 mikroBASIC PRO for AVR® 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

* Funciones de la librería CANSPI usadas en el programa

Otras funciones mikroBASIC PRO for AVR usadas en el programa:
 Delay_us()
 SPI1_init()
 SPI1_read()

GO TO

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

