

Oké. Nu hebt u... een CAN-bus nodig



Vaak moeten binnen een systeem meerdere microcontrollers verschillende bewerkingen uitvoeren om ze als een geheel te laten functioneren. Het volgende voorbeeld laat zien hoe drie microcontrollers worden aangesloten op een CAN-netwerk en hoe met filters in CAN-knooppunten berichten kunnen worden gefilterd.

EasyAVR5A ontwikkelsystemen en CAN-SPI modules

Door: Zoran Ristic
MikroElektronika - Software Department

Altijd wanneer meerdere randapparaten een gemeenschappelijke databus gebruiken, moet worden vastgelegd hoe de bus moet worden gebruikt. Daarvoor wordt in de industrie vrij algemeen het CAN-protocol gevolgd, waarin nauwkeurig is vastgelegd hoe meerdere apparaten in een netwerk moeten worden aangesloten. Het protocol definieert voornamelijk de prioriteiten bij het gebruik van de bus en lost "conflictsituaties" in de hardware op, in het geval dat meer randapparaten tegelijk proberen te communiceren.

Hardware

In het hier beschreven voorbeeld is een CAN-bus zo geconfigureerd dat het eerste apparaat berichten uitzendt die 0x10 en 0x11 als ID hebben, terwijl het tweede en derde apparaat berichten uitzenden met respectievelijk 0x12 en 0x13 als ID. De CAN-knooppunten zijn zo ingesteld dat het tweede knooppunt uitsluitend reageert op inkomende berichten met 0x10 als ID. Het derde reageert alleen op berichten met 0x11 als ID. Het eerste apparaat staat dus ingesteld op ontvangst van berichten met ID 0x12 en ID 0x13 (zie figuur 2). Berichten laten zich gemakkelijk filteren door de routine CANSPISetFilter aan te roepen die tevens alle noodzakelijke instellingen van de microcontroller-registers en de CAN SPI-print afhandelt.

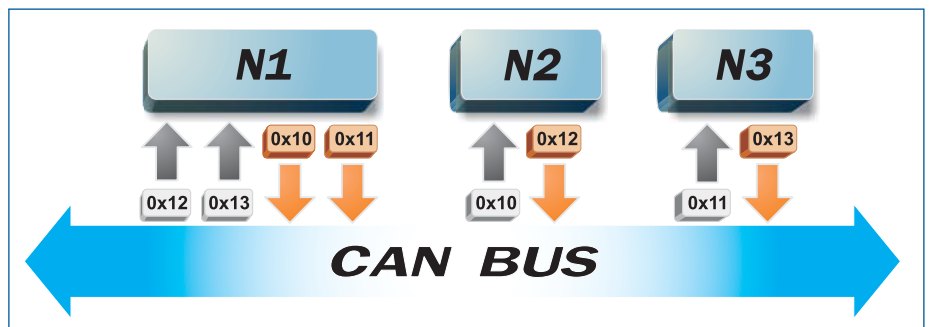
In het algemeen hoeft volgens het CAN-protocol geen master-apparaat op de bus aangesloten te zijn. Ter verduidelijking van het hier beschreven voorbeeld is, met behoud van de algemene werkwijze, het eerste apparaat echter zo ingesteld dat het de communicatie over het netwerk controleert terwijl de beide andere apparaten op afzonderlijke oproepen reageren.

Software

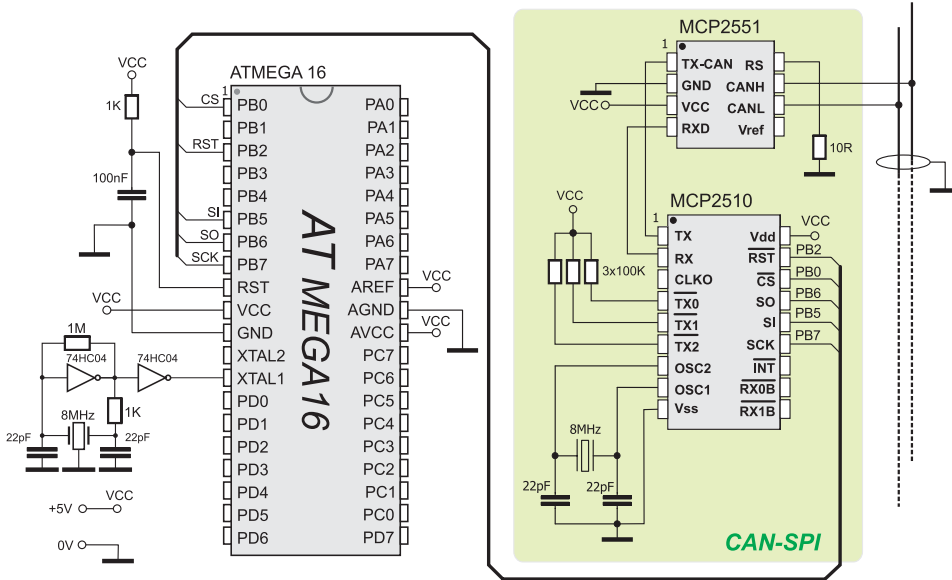
Bij het verzenden van een bericht geeft het master-knooppunt het aangeroepen knooppunt voldoende tijd om te reageren. Reageert een elders opgesteld knooppunt niet binnen de vereiste tijd, dan rapporteert de master een fout in het huidige bericht en vervolgt het het aanroepen van andere knooppunten (zie figuur 3). In het geval dat een randapparaat op een CAN-knooppunt op hetzelfde moment reageert als een

ander knooppunt, is er sprake van een "conflictsituatie" op de CAN-bus. In dat geval bepalen echter de prioriteit van het apparaatadres en het CAN-protocol dat het knooppunt dat het bericht met de laagste prioriteit uitzendt, zich terugtrekt van de bus zodat het knooppunt dat een bericht met hogere prioriteit uitzendt zijn uitzending onmiddellijk kan vervolgen.

Zoals al eerder werd opgemerkt, wordt een interne SPI-module van de microcontroller gebruikt om data over de CAN-bus te verzenden. Enkele voordelen van het gebruik van een interne SPI-module van de microcontroller zijn: de mogelijkheid tijdens het verzenden en ontvangen van data interrupts te genereren; de SPI-module werkt onafhankelijk van andere randapparaten en heeft een eenvoudige configuratie. Met de functies van de CAN SPI-bibliotheek kunnen de bedrijfsmodus van het CAN-netwerken en de knooppuntfilters worden



Figuur 1. Filteren van berichten



Schema 1. Aansluiten van de CAN-SPI-module op een ATmega 16

ingesteld, kunnen data van de CAN SPI board buffer worden gelezen enzovoort. LED's op de aansluitpennen van de microcontroller geven aan of het netwerk op de juiste wijze functioneert. Wanneer knooppunt 2 op een aanroep van knooppunt 1 reageert, lichten automatisch de PORTB-LED'S op. Reageert knooppunt 3 op een aanroep, dan lichten de PORTD-LED's op. Bij dit voorbeeld is ook de broncode voor alle drie de knooppunten in het netwerk beschikbaar. Om voor al deze knooppunten een afzonderlijk HEX-bestand te maken, hoeft maar één DEFINE-directive in de header van het voorbeeld te worden geschreven.

Programma ter illustratie van de werking van een CAN-netwerk

```

program CanSpi;
{ Description: This program demonstrates how to make a CAN
network using mikroElektronika
CANSPI boards and mikroPascal compiler.
Target device: Microchip ATMEL 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
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;
// Error flag
// 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
PORTD := 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_FHSBG2_FRG_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);
// 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
RxTx_Data[0] := 0x40;
// set initial data to be sent

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

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

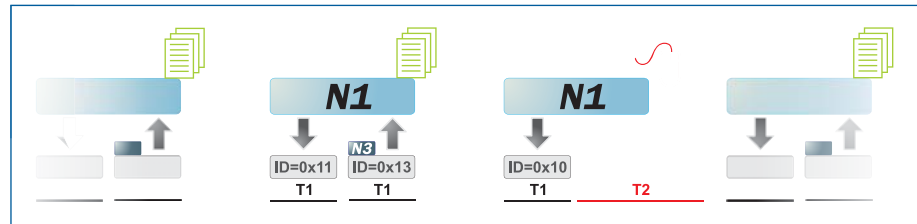
while (TRUE) do
// endless loop
begin
Msg_Rcvd := CANSPIRead(Rx_ID, RxTx_Data, Rx_Data_Len,
Can_Rcv_Flags); // attempt receive message
if (Msg_Rcvd) then begin
// if message is received then check id

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

inc(ErrorCount);
// increment error indicator
Delay_ms(10);
// wait for 10ms
if (ErrorCount > 10) then begin
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, RxTx_Data, 1, Can_Send_Flags);
// send new message
end;

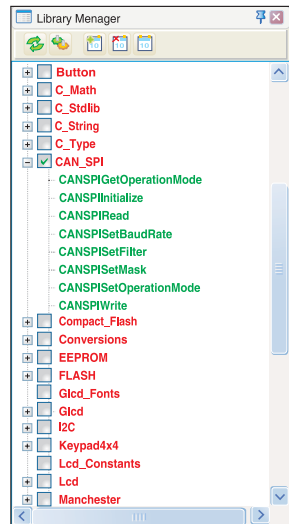
end;
end;
end.
    
```



Figuur 2. Een voorbeeld van communicatie

Samenvattend kan worden gesteld dat hier niet alleen een manier is beschreven waarop microcontrollers op een CAN-bus kunnen worden aangesloten, maar ook hoe met behulp van een communicatieprotocol fouten kunnen worden gedetecteerd in het geval dat een knooppunt elders niet reageert zoals verwacht, de wijze waarop berichten met behulp van CAN-filters kunnen worden gefilterd en hoe de communicatie over een CAN-bus in het algemeen tot stand komt.

mikroPASCAL PRO for AVR @ bibliotheek-editor met gebruiksklare bibliotheken, zoals: CAN_SPI, GLCD, Ethernet enzovoort.



In het programma gebruikte functies

- CANSPIGetOperationMode() Huidige bedrijfsmodus
 - CANSPIInitialize()* CANSPI-module initialiseren
 - CANIRead()* Bericht lezen
 - CANSPISetBaudRate() CANSPI-baudrate instellen
 - CANSPISetFilter()* Berichtenfilter configureren
 - CANSPISetMask()* Geavanceerd filteren configureren
 - CANSPISet OperationMode()* Huidige bedrijfsmodus
 - CANSPIWrite()* Bericht schrijven
- * In het programma gebruikte CANSPI-bibliotheekfuncties
- Andere in het programma gebruikte mikroPASCAL PRO for AVR-functies:
- Delay_us()
 - SPI1_init()
 - SPI1_read()

GO TO De voor dit voorbeeld in C, Basic en Pascal voor AVR® microcontrollers geschreven code vindt u, evenals de voor PIC®, dsPIC® en 8051 microcontrollers geschreven programma's, op onze website: www.mikroe.com/en/article/

