

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.

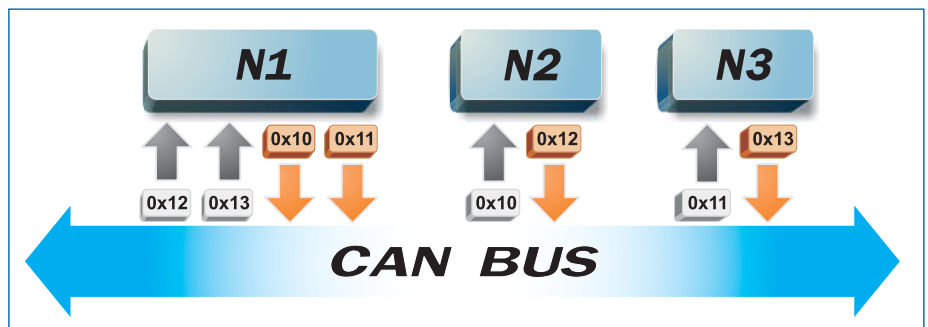
In het algemeen hoeft volgens het CAN-protocol geen master-apparaat op de bus aangesloten te zijn. Ter verduidelijking van het hier beschreven voorbeeld is, met behoud van de algemene werkwijze, het eerste apparaat echter zo ingesteld dat het de communicatie over het netwerk controleert terwijl de beide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

Software

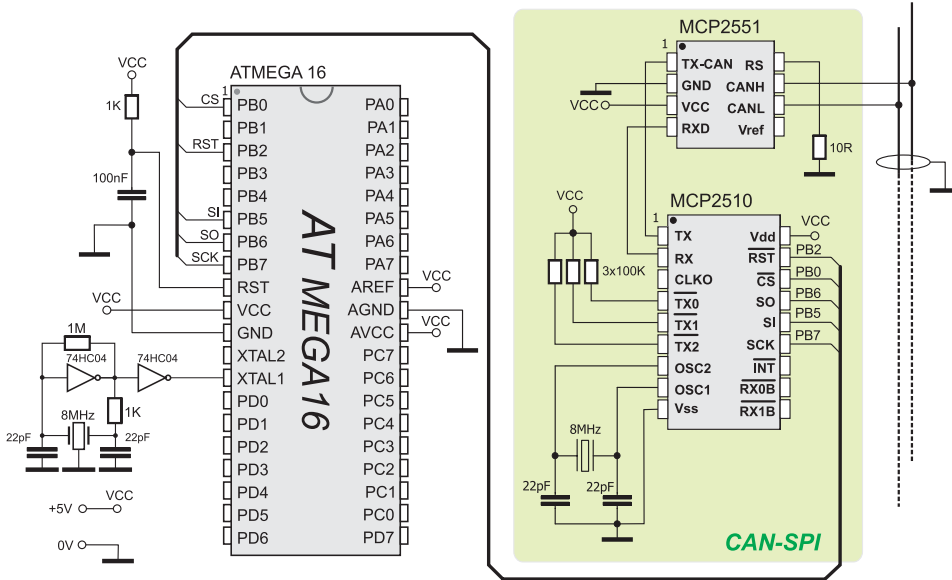
Bij het verzenden van een bericht geeft het master-knooppunt het aangeroepen knooppunt voldoende tijd om te reageren. Reageert een elders opgesteld knooppunt niet binnen de vereiste tijd, dan rapporteert de master een fout in het huidige bericht en vervolgt het het aanroepen van andere knooppunten (zie 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

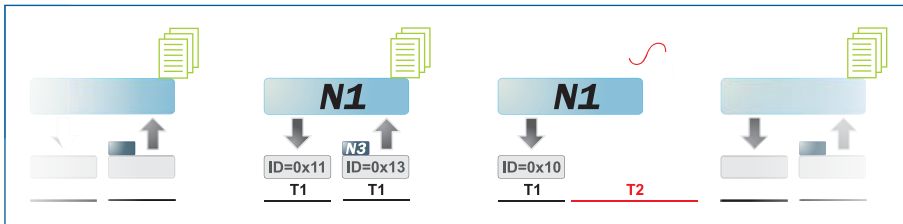


Figuur 1. Filteren van berichten



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

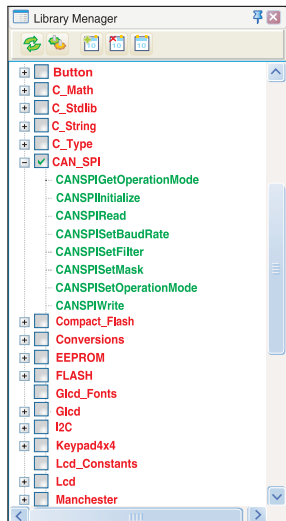
ingesteld, kunnen data van de CAN SPI board buffer worden gelezen enzovoort. LED's op de aansluitpennen van de microcontroller geven aan of het netwerk op de juiste wijze functioneert. Wanneer knooppunt 2 op een aanroep van knooppunt 1 reageert, lichten automatisch de PORTB-LED'S op. Reageert knooppunt 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.



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.

mikroC 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
- CANSPISetOperationMode()* Huidige bedrijfsmodus
- CANSPIWrite()* Bericht schrijven

* In het programma gebruikte CANSPI-bibliotheekfuncties

- Ander in het programma gebruikte mikroC PRO for AVR-functies:
- Delay_us()
- SPI1_init()
- SPI1_read()

Programma ter illustratie van de werking van een CAN-netwerk

```

char Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags;
// can flags
char Rx_Data_Len;
// received data length in bytes
char RxTx_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 PORTB.B0;
// Chip select (CS) pin for CANSPI board
sbit CanSpi_CS_Direction at DDRB.B0;
// Direction register for CS pin
sbit CanSpi_Rst at PORTB.B2;
// Reset pin for CANSPI board
sbit CanSpi_Rst_Direction at DDRB.B2;
// Direction register for Reset pin
// End CANSPI module connections
void main(){
ADCSRA.B7 = 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 &
// form value to be used
_CANSPI_TX_XTD_FRAME &
// with CANSPIWrite
_CANSPI_TX_NO_RTR_FRAME;

Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE &
// form value to be used
_CANSPI_CONFIG_PHASE2_PRG_ON &
// with CANSPIInit
_CANSPI_CONFIG_XTD_MSG &
_CANSPI_CONFIG_DBL_BUFFER_ON &
_CANSPI_CONFIG_VALID_XTD_MSG;

SPI_Init();
Spi_Rd_Ptr = SPI1_Read;
// 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_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 (1)
// endless loop
{
Msg_Rcvd = CANSPIRead(&Rx_ID, RxTx_Data, &Rx_Data_Len,
&Can_Rcv_Flags); // attempt receive message
if (Msg_Rcvd) {
// if message is received then check id

if (Rx_ID == 0x12)
// 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
Tx_ID++;
// switch to next message
if (Tx_ID > 0x11) Tx_ID = 0x10;
// check overflow
}
else {
// an error occurred, wait for a while

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, RxTx_Data, 1, Can_Send_Flags);
// send new message
}
}
}
}
    
```

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/

