

AVRco Library für CAN-SPI-Adapter

Einfaches senden und Empfangen von CAN-Meldungen mit einem AVR Mikrocontroller.

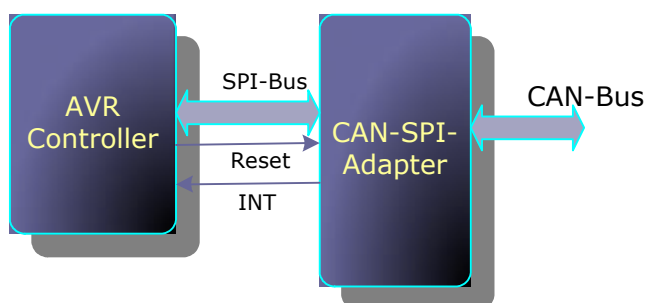
Einführung

Der MCP2515 von Microchip ist ein Standalone-CAN-Controller, der via SPI-Bus mit einem Host-Mikrocontroller kommuniziert. Die hier beschriebene Software stellt einige grundlegende Funktionen zur Verfügung, die es dem Anwender erlauben, auf einfache Weise CAN-Meldungen mit einem AVR-Mikrocontroller zu versenden und zu empfangen. Die Funktionen sind in einer vorkompilierten Unit für das AVRco-Entwicklungssystem zusammengefasst.

Systemvoraussetzungen

- AVR Mikrocontroller
- CAN-SPI-Adapter mit MCP2515
- Pascal compiler von [E-Lab \(www.e-lab.de\)](http://www.e-lab.de), Professional-Version [3].

Hardware-Übersicht



Interface-Beschreibung

Konstante

Meldungsstruktur

```
const
  CAN_MSG_LENGTH:   Byte = 5;
  CAN_DATA_LENGTH:  Byte = 8;
  RX_QUEUE_SIZE:    Byte = 3;
  TX_QUEUE_SIZE:    Byte = 3;
```

Returncodes

```
RC_SUCCESS:        Byte = 0;
RC_QUEUEEMPTY:     Byte = 1;
RC_QUEUEFULL:      Byte = 1;
RC_CANNOTRECEIVE: Byte = 2;
RC_CANNOTTRANSMIT: Byte = 2;
RC_PARAMERROR:     Byte = 3;
```

Weitere exportierte Konstante sind in der Datei mcp2515.inc enthalten. Diese Datei wird bereits von der Treiber-Unit importiert und muss deshalb vom Anwenderprogramm nicht importiert werden.

Exportierte Datentypen

CAN_MESSAGE

Mit der Datenstruktur CAN_MESSAGE werden die CAN-Meldungen zwischen Applikation und Driver-Unit ausgetauscht.

```
CAN_DATA      = array [0..CAN_DATA_LENGTH] of Byte;
CAN_MESSAGE = record
  SID          : Word;
  EID          : LongWord;
  ExtendedId   : Boolean;
  RTR          : Boolean;
  SRR          : Boolean;
  DataLength   : Byte;
  Data         : CAN_DATA;
end;
tMessagePtr = Pointer to CAN_MESSAGE;
```

CAN_DATA enthält die Nutzdaten.

SID enthält den Standard-Identifizier (untere 11Bit)

EID enthält den Extended Identifizier (untere 29Bit)

SID und EID werden vom Driver auf die entsprechenden Register des MCP2515 umgesetzt.

ExtendedId ist true, wenn der EID benutzt werden soll

RTR ist true, wenn die Meldung eine Abfrage-Meldung ist

SRR ist true, wenn ein Standard Frame Remote Transmit Request empfangen wurde

DataLength enthält die Anzahl Bytes in CAN_DATA (0..8 Bytes)

Globale Variablen

Puffer für empfangene und zu sendende Meldungen

Die Driver-Unit deklariert zwei Pipes als globale Variablen, über welche die CAN-Meldungen ausgetauscht werden.

```
var
  RXQueue : Pipe [RX_QUEUE_SIZE] of CAN_MESSAGE;
  TXQueue : Pipe [TX_QUEUE_SIZE] of CAN_MESSAGE;
```

Callback Funktionen

SPI-Kommunikation

In der Treiber-Unit ist die Kommunikation via SPI mit dem MCP2515 gekapselt. Damit der Anwender einen beliebigen Port-Pin für die Select-Leitung des MCP2515 verwenden kann, müssen die folgenden Callback-Prozeduren im Anwenderprogramm definiert werden.

```
Procedure SELECT_MCP;
Procedure UNSELECT_MCP;
```

Im Hauptprogramm müssen die folgenden Variablen entsprechend gesetzt werden:

```
SelectMCP:= @SELECT_MCP;
UnselectMCP:= @UNSELECT_MCP;
```

Ein Beispiel findet sich im Kapitel „Verwendung der Treiber-Unit“.

Funktionen für Registerzugriff

MCP_GetStatus

```
Function MCP_GetStatus: Byte;
```

Diese Funktion führt im MCP2515 den ReadStatus-Befehl aus. Der einzelnen Bits im Rückgabewert haben folgende Bedeutung:

Bit Nr.	Flag	Bedeutung wenn Bit gesetzt
0	CANINTF.RX0IF	Interrupt-Flag: Empfangs-Buffer 0 voll
1	CANINTFL.RX1IF	Interrupt-Flag: Empfangs-Buffer 1 voll
2	TXB0CNTRL.TXREQ	Sende-Buffer 0: Sendung pendent
3	CANINTF.TX0IF	Interrupt-Flag: Sende-Buffer 0 leer
4	TXB1CNTRL.TXREQ	Sende-Buffer 1: Sendung pendent
5	CANINTF.TX1IF	Interrupt-Flag: Sende-Buffer 1 leer
6	TXB2CNTRL.TXREQ	Sende-Buffer 2: Sendung pendent
7	CANINTF.TX2IF	Interrupt-Flag: Sende-Buffer 2 leer

Weitere Informationen in [1], Fig. 11-8

MCP_GetError

```
Function MCP_GetError: Byte;
```

Diese Funktion liest im MCP2515 das Error-Flag-Register aus. Der einzelnen Bits im Rückgabewert haben folgende Bedeutung:

Bit Nr.	Flag	Bit ist gesetzt wenn...
0	EWARN	der Fehlerzähler für Senden oder Empfangen den Wert 96 überschreitet.
1	RXWAR	der Fehlerzähler für Empfangen den Wert 96 überschreitet.
2	TXWAR	der Fehlerzähler für Senden den Wert 96 überschreitet.
3	RXEP	der Fehlerzähler für Empfangen den Wert 128 überschreitet.
4	TXEP	der Fehlerzähler für Senden den Wert 128 überschreitet.
5	TXBO	der Fehlerzähler für Senden den Wert 255 erreicht.
6	RX0OVR	Eine gültige Meldung im Empfangsbuffer 0 empfangen wurde und CANINTF.RX0IF = 1 ist
7	RX1OVR	Eine gültige Meldung im Empfangsbuffer 1 empfangen wurde und CANINTF.RX1IF = 1 ist

Weitere Informationen in [1], Kap. 6

SetAddressFilter

```
Procedure SetAddressFilter (Address: Byte; Data: Byte);
```

Diese Prozedur setzt das Filter- oder das Mask-Register an der angegebenen Adresse auf den angegebenen Wert. Dazu wird der MCP2515 in den

Konfigurations-Modus gesetzt und nach dem Setzen des Filters wieder in der Normal-Modus.

Details zur Definition der Adress-Filter in [1], Kap. 4.5.

MCP_WriteReg

```
Procedure MCP_WriteReg (Address: Byte; Data: Byte);
```

Diese Prozedur schreibt den angegebenen Wert in das Register des MCP2515 an der angegebenen Adresse.

MCP_ModifyReg

```
Procedure MCP_ModifyReg (Address: Byte; Mask: Byte; Data: Byte);
```

Diese Prozedur modifiziert die mit „Mask“ definierten Bits im Register an der angegebenen Adresse mit den angegebenen Daten.

Meldungen senden und empfangen

TransmitMessages

```
Function TransmitMessages: Byte;
```

Diese Funktion sendet soviele Meldungen, wie in der Sende-Pipe vorhanden sind. Falls eine Meldung nicht abgesetzt werden kann, wird im Rückgabewert ein Fehlercode übergeben.

RC_SUCCESS: Meldungen erfolgreich abgesetzt.

RC_CANNOTTRANSMIT: alle Sendebuffer des MCP2515 sind besetzt.

RC_QUEUEEMPTY: Sende-Pipe ist leer.

Während der Ausführung dieser Funktion sind die Interrupts gesperrt, da die Funktion vom Interrupt-Handler aufgerufen werden kann.

ReceiveMessages

```
Procedure ReceiveMessages;
```

Diese Prozedur wird aufgerufen, wenn der MCP2515 einen Interrupt auslöst oder in der Polling-Routine.

Wenn Meldungen im Empfangsbuffer des MCP2515 sind, werden diese ausgelesen, in die definierte Datenstruktur gewandelt und dann in die Empfangs-Pipe übergeben.

Während der Ausführung dieser Funktion sind die Interrupts gesperrt, da die Funktion vom Interrupt-Handler aufgerufen werden kann.

CAN-Adapter Initialisieren

CAN_Init

```
Procedure CAN_Init (CNF1, CNF2, CNF3: Byte);
```

Diese Prozedur muss vom Anwenderprogramm aufgerufen werden, bevor der MCP2515 angesprochen wird.

Hier werden als erstes der MCP2515 zurückgesetzt und dann die Konfigurationsregister initialisiert. Der Inhalt dieser Konfigurationsregister ist entsprechend den Angaben im Datenblatt des MCP2515 zu setzen (Kap. 5.0 – Bit Timing). Ein Beispiel findet sich auch im Kapitel „Verwendung der Treiber-Unit“.

Hilfs-Konstante zum Einstellen dieser Parameter sind in der Datei mcp2515.inc enthalten.

Anschliessend wird der MCP2515 in den NORMAL-Modus gesetzt.

CAN_Setup

```
Procedure CAN_Setup (InterruptMode: Boolean);
```

Wenn der Parameter „true“ ist, werden hier die Interrupts des MCP2515 freigegeben. Globale Interrupts und der entsprechende externe Interrupt des Host-Controllers müssen vom Anwenderprogramm aus freigegeben werden.

Defaultmässig, also ohne Aufruf dieser Prozedur, ist der Polling-Betrieb gewählt.

**Interrupt-
Behandlung*****MCP_Interrupt***

```
Procedure MCP_Interrupt;
```

Diese Prozedur muss vom Interrupt-Handler des Anwenderprogramms aufgerufen werden. In ihr werden ReceiveMessages und TransmitMessages aufgerufen. Bei letzterem wird der Rückgabewert nicht ausgewertet.

Alternativ kann der Interrupt-Handler diese beiden Prozeduren auch direkt aufrufen.

Verwendung der Treiber-Unit

Im Applikationsprogramm sind folgende Eintragungen zu machen:

Allgemein

Deklarationsteil

1. SPI-Driver von AVRco einbinden:

```
Import SPIdriver;           // und weitere Imports, je nach Applikation
From System Import Pipes; // und weitere Imports, je nach Applikation
Define
    // applikationsspezifische, dann...
    SPIpresc      = 0;
    SPIOrder      = MSB;
    SPICPOL       = 0;
    SPICPHA       = 0;
    SPI_SS        = false;
```

2. Unit einbinden. Gleich nach den Define-Statements folgenden Eintrag:

```
uses CANmessages;
```

Im Impelentationsteil

3. Konstante für die Konfigurationsregister des MCP2515 definieren

```
const
    CNF1:          Byte = SJW1 + 3;
    CNF2:          Byte = (BTLMODE + (7-1)*8 + (2-1));
    CNF3:          Byte = (SOF_DISABLE + WAKFIL_DISABLE + (6-1));
```

4. Select-Pin des MCP2515 festlegen. Im folgenden wird als Beispiel Bit 0 von PortB verwendet.

```
var
    MCP_SELECT [@PortB, 0]: Bit;
```

5. In der Port-Initialisierung den Select-Pin als Ausgang definieren

```
procedure InitPorts;
begin
    // weitere Portpins gemäss Anwendung
    DDRB.0:= 1;
end InitPorts;
```

6. Callback-Prozeduren für die Selektierung des MCP2515 bereitstellen

```
Procedure SELECT_MCP;
Begin
    MCP_SELECT:= 0;
End;
Procedure UNSELECT_MCP;
Begin
    MCP_SELECT:= 1;
End;
```

Variante: Interrupt-Betrieb

Im Implementationsteil

7. Interrupt-Freigabe. In diesem Beispiel wird INT4 des ATmega128 verwendet. Je nach Anschluss des INT-Ausgangs des Adapters sind die entsprechenden Bits zu setzen.

```
Procedure EnableCANint;
begin
    incl (EICR, 1);
    excl (EICR, 0);           // falling edge INT4
```

```

    excl (EIFR, 4);           // reset    INT4
    incl (EIMSK, 4);        // enable  INT4
end EnableCANint;

```

8. Interrupt-Handler bereitstellen

```

Interrupt INT4; // je nach Anschluss des INT-Ausgangs des Adapters
Begin
    TransmitMessages;
    ReceiveMessages;
End;

```

Im Hauptprogrammteil

```

var
    msg:    CAN_MESSAGE_STRUCT;
begin
    SelectMCP:= @SELECT_MCP;
    UnselectMCP:= @UNSELECT_MCP;
    InitPorts;
    EnableInts;
    CAN_Init;
    CAN_SETUP (true);
    EnableCANint;
    loop
        if PipeStat (RXqueue) > 0 then
            PipeRecv (RXqueue, msg);
            // empfangene Meldung auswerten
        endif;
    endloop;
end.

```

Variante: Polling-Betrieb

Im Hauptprogrammteil

```

var
    msg:    CAN_MESSAGE_STRUCT;
begin
    SelectMCP:= @SELECT_MCP;
    UnselectMCP:= @UNSELECT_MCP;
    InitPorts;
    EnableInts;
    CAN_Init;
    CAN_SETUP (false);
    loop
        ReceiveMessages;
        TransmitMessages;
        if PipeStat (RXqueue) > 0 then
            PipeRecv (RXqueue, msg);
            // empfangene Meldung auswerten
        endif;
    endloop;
end.

```

**Weitere
Informationen**

- [1] Microchip Technology Inc., Datenblatt MCP2515, Doc. No. DS21801B
- [2] Website des Herstellers: www.avrcard.com
- [3] Entwicklungssystem AVRco von E-Lab Computers, www.e-lab.de

**Kontakt und
Support**

Anfragen können an folgende Adresse gerichtet werden:

Elektronik-Atelier Kallen

Steinackerweg 14
CH-3075 Rüfenacht / Schweiz

Web: www.avrcard.com

Email: info@avrcard.com

Tel: +41 31 832 1441

Fax: +41 31 832 1442