Application Note AN062

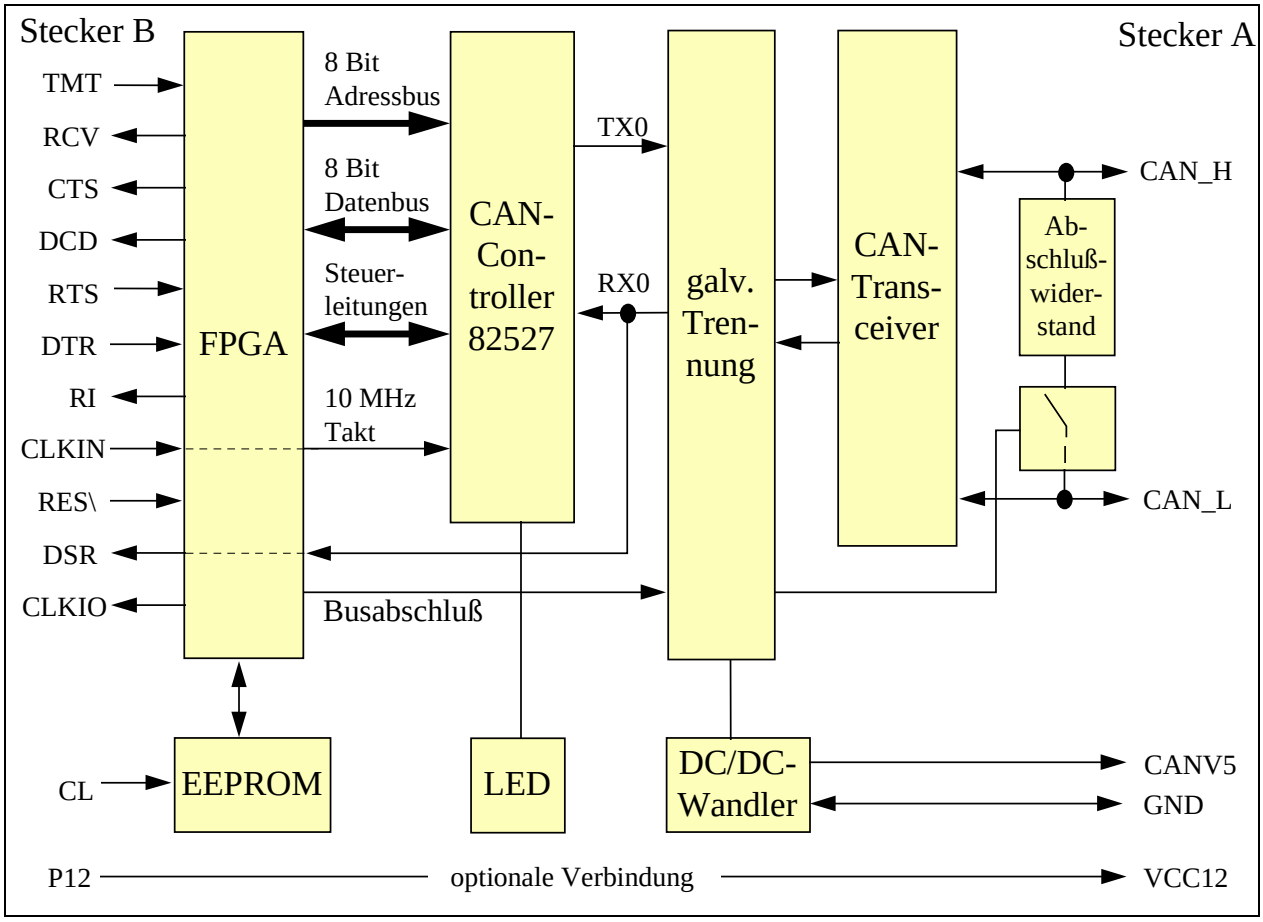
Kommunikation mit dem S-Link SL-CANi

Autor: JD Datei: AN062.DOC (11 Seiten)

In diesem Dokument wird beschrieben, wie die Kommunikation einer seriellen Schnittstelle mit dem S-Link SL-CANi funktioniert. Für den Einsatz des S-Links auf der Multi-COM Karte von SORCUS sind diese Informationen nicht relevant, da das S-Link zusammen mit einem Treiberprogramm ausgeliefert wird. Weitere Informationen über den allgemeinen Aufbau der S-Links stehen in der SORCUS Application Note AN050.

Aufbau des S-Links SL-CANi

Die Kommunikation mit dem CAN-Controller wird durch ein FPGA gesteuert. Durch dieses kann auf die Register des CAN-Controllers Intel 82527 zugegriffen werden.



Die Kommunikation zwischen der Trägerkarte und dem FPGA des S-Links geschieht seriell über die Leitungen TMT und RCV sowie einige Steuersignale. Es kann zwischen verschiedenen Übertragungsformen gewählt werden:

- Asynchrone Kommunikation mit einer Übertragungsrate von 9600 Bit/s bis 1,25 MBit/s, wobei die gewählte Baudrate vom FPGA automatisch erkannt wird.
- Synchrone Kommunikation mit 1,25 MHz oder 3,33 MHz. Der Takt wird vom FPGA am S-Link Stecker B, Pin 19 (CLKIO) ausgegeben.

In beiden Fällen sind die Übertragungsparameter 8 Datenbit, keine Parität, 1 Stopbit (8, n, 1) zu wählen. Die angegebenen Werte gelten für einen Eingangstakt von 10 MHz am S-Link.

Nähere Angaben zum CAN-Controller 82527 finden sich in den Datenblättern von Intel (Intel-Bestell-Nummer 272250-006 und 272410-002). Auf dem SL-CANi wird dieser im Mode 3 betrieben ('8-Bit Non-Multiplexed Asynchronous Mode'). Als Taktsignal bezieht er den 10 MHz Takt vom FPGA. Alternativ kann auch ein 16 MHz Quarz bestückt werden, der dann das Taktsignal für den CAN-Controller bereitstellt.

Die Interrupt-Leitung des CAN-Controllers geht durch das FPGA an die RI-Leitung. Über die Bits 1 und 2 von Port 2 (Registeradresse EFh) des CAN-Controllers kann die Auslösung der Interrupts eingestellt werden:

Port 2, Bit 1	**Port 2, Bit 2**	**Interrupt-Leitung RI**
0	0	CAN-Controller Interruptleitung wird invertiert weitergegeben
0	1	RI = 0
1	0	RI = 1
1	1	CAN-Controller Interruptleitung wird ohne Invertierung durchgegeben

Die Ansteuerung der LED erfolgt über Bit 0 von Port 2 des CAN-Controllers (0=LED ein). Port 2 des CAN-Controllers ist als Ausgang zu konfigurieren (Registeradresse AFh).

Der Zustand des Busabschlußwiderstandes wird im FPGA gespeichert und über einen Ausgang des FPGA´s ausgegeben. Die Ansteuerung des schaltbaren Widerstands erfolgt galvanisch getrennt.

Der Zustand der CAN-Empfangsleitung RX0 kann jederzeit zu Prüfzwecken vom Pin DSR gelesen werden.

Bit 2 und Bit 0 im CPU-Interface-Register (Registeradresse 02) des CAN-Controllers sind =0 zu setzen. Das Bus Configuration Register (Registeradresse 2Fh) ist mit dem Wert 4Ah zu initialisieren.

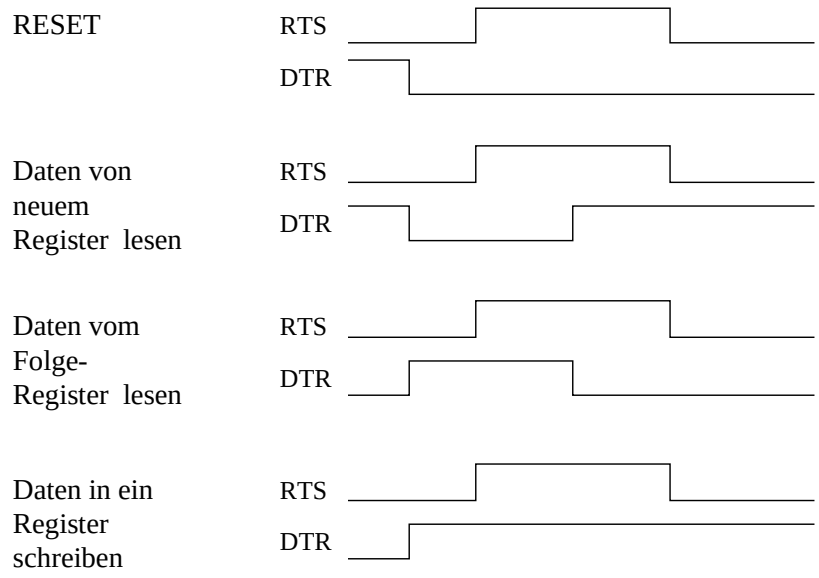
FPGA Design

Im FPGA stehen für die Kommunikation mit dem S-Link 4 Befehle zur Verfügung:

- RESET des S-Links (FPGA und CAN-Controller)
- Daten aus einem Register des CAN-Controllers lesen (die Registernummer muß an das FPGA gegeben werden).
- Daten aus dem folgenden Register des CAN-Controllers lesen (ohne Übertragung einer neuen Registernummer)
- Daten in ein Register des CAN-Controllers schreiben

Ein Schreib- oder Lesebefehl erfordert im Normalfall die Registernummer (0 bis 255) im CAN-Controller sowie Daten (entweder hin oder zurück). Die Registernummer wird im FPGA zwischengespeichert und nach der Befehlsausführung um eins erhöht, so daß beim nächsten Zugriff die Registernummer nicht zum FPGA gesendet werden muß, wenn auf das Folgeregister zugegriffen werden soll.

Die Codierung der Befehle geschieht über die beiden Steuerleitungen DTR und RTS. Entscheidend ist der Zustand des DTR-Eingangs bei der positiven und der negativen Flanke an RTS. DTR muß mindestens 20 ns vor der Flanke an RTS stabil anliegen. Die Befehlsausführung beginnt bei negativer Flanke an RTS.



Das FPGA liefert über die CTS-Leitung eine Bestätigung der korrekten Befehlsausführung, indem die CTS-Leitung ihren Zustand ändert. Eine Zustandsänderung der DCD-Leitung signalisiert vorher bereits, daß das FPGA bereit für einen neuen Befehl ist.

Die Daten werden seriell über die Leitung TMT an das SL-CANi gesendet und über RCV empfangen.

Reset und Initialisierung

Ein Reset des FPGA's und des CAN-Controllers erfolgt wie oben gezeigt. Die CTS Leitung geht durch das Reset zunächst auf 0. Wenn es anschließend auf 1 geht, zeigt es die Betriebsbereitschaft des SL-CANi an. Das FPGA befindet sich danach zunächst in der asynchronen Kommunikations-Betriebsart. Die Initialisierung muß deshalb asynchron geschehen.

Das Senden der Bytes 80h und 55h dient der automatischen Baudratenerkennung im FPGA. Ist diese erfolgreich verlaufen, geht CTS wiederum auf 0.

Durch Senden des folgenden Bytes wird das S-Link konfiguriert:

Bit	**Bedeutung**
0	=1: zeigt das Ende der Initialisierung an =0: nicht zulässig
1	=0: die Kommunikation mit dem FPGA erfolgt weiterhin asynchron =1: die Kommunikation mit dem FPGA erfolgt synchron
2	=0: Synchrone Übertragungsrate 1,25 MHz =1: Synchrone Übertragungsrate 3,33 MHz
3	=0: Busabschlußwiderstand aus =1: Busabschlußwiderstand ein
4 bis 7	reserviert

Der Empfang dieses Konfigurationsbytes wird durch das Umschalten von CTS auf 1 bestätigt. Ab diesem Zeitpunkt erfolgt die Kommunikation mit dem S-Link im soeben eingestellten Mode. Wurde die synchrone Betriebsart gewählt, ist der serielle Schnittstellen-Controller jetzt umzuprogrammieren.

Im nächsten Schritt wird die FPGA-Versionsnummer ausgelesen. Dazu wird mit DTR und RTS der Befehl "Daten vom Folge-Register lesen" (s.o.) an das FPGA gesendet. Das Antwortbyte enthält in den unteren 4 Bit die Versionsnummer, in den oberen 4 Bit das Testmuster 0101. Das Umkippen der CTS-Leitung auf 0 zeigt die Beendigung der Initialisierung an.

EEPROM-Zugriffe

Zugriffe auf das S-Link-EEPROM erfolgen über die Anschlüsse RTS (EEPROM-Dateneingang), DSR (EEPROM-Datenausgang), DTR (EEPROM-Takteingang) und CL (Select-Eingang). Diese Signale werden ansonsten für die Befehlsverarbeitung benutzt. Ihre Zustände müssen daher vor einem EEPROM-Zugriff gesichert werden, um sie anschließend wiederherstellen zu können. Die Zustände der beiden Ausgänge RTS und DTR können - sobald CL aktiviert (=1) ist - an CTS (für RTS) und DCD (für DTR) zurückgelesen werden. Weitere wichtige Hinweise zu EEPROM-Zugriffen finden sich in der Application Note AN050.

Optimierungsmöglichkeiten

Um den Kommunikationsablauf zu beschleunigen, wurden im FPGA einige Möglichkeiten zur Optimierung implementiert. Welche davon in der Software implementiert werden, hängt vom gewünschten Software-Aufwand und der Leistungsfähigkeit der seriellen Schnittstelle (inkl. Steuerleitungen) ab, die mit dem S-Link kommuniziert.

Verwenden der Folgeadresse

Bei den Lese- und Schreibbefehlen wird die angesprochene CAN-Controller Registeradresse jeweils im FPGA zwischengespeichert und nach Beendigung des Befehls um eins erhöht. Soll im folgenden Befehl auf dieses Register zugegriffen werden, braucht keine neue Adresse übertragen zu werden. Bei solch einem Folgezugriff wird die zwischengespeicherte Adresse ebenfalls um 1 inkrementiert.

Lesen und Schreiben von Registern ohne Befehl durch DTR und RTS

Folgt ein *Schreibbefehl* auf einen anderen Schreibbefehl, und bezieht sich dieser auf die im FPGA zwischengespeicherte Adresse, braucht kein Befehl mit DTR und RTS gesendet zu werden. Es genügt in diesem Fall, die Daten an das FPGA zu senden. Diese werden automatisch in die zwischengespeicherte Registeradresse des CAN-Controllers geschrieben. Die zwischengespeicherte Adresse wird wiederum um 1 inkrementiert.

Folgt ein *Lesebefehl* auf einen anderen Lesebefehl, ist keine Einleitung des Befehls über DTR und RTS erforderlich. Statt dessen kann die gewünschte Registeradresse an das FPGA gesendet und anschließend der Registerinhalt abgeholt werden. Bei einem Lesezugriff auf die Folgeadresse, besteht die Auswahl zwischen zwei Möglichkeiten:

- Befehl "Daten vom Folgeregister lesen" ohne Übertragung der Adresse
- Übertragen der Adresse ohne vorherigen Befehl über DTR und RTS

Welche dieser beiden Varianten die günstigere ist, hängt vom Anwendungsfall ab.

Parallelbearbeitung zweier Befehle

Es ist möglich, während der Ausführungsphase eines Befehls einen neuen Befehl inklusive Registeradresse und Datenbyte zum FPGA zu senden. Dieser wird dort zwischengespeichert und ausgeführt, sobald der alte Befehl vollständig abgearbeitet worden ist.

Ein Toggeln der DCD-Leitung zeigt an, daß der Zwischenspeicher im FPGA frei für einen neuen Befehl ist (der vorherige Befehl ist in Bearbeitung). Erst dann darf ein neuer Befehl an das S-Link gesendet werden. Daraufhin muß auf das zweite Toggeln von DCD gewartet werden. Dieses signalisiert, daß auch der zweite Befehl vom FPGA an den CAN Controller weitergereicht wurde. Daraus kann geschlossen werden, daß die Abarbeitung des ersten Befehls fertig ist (dieses ließe sich auch am zwischenzeitlichen Toggeln von CTS ablesen).

Es können weitere Befehle gesendet werden, zwischen denen jeweils nur auf das Toggeln von DCD gewartet werden muß. Auf die Zustandsänderung von CTS muß lediglich am Ende der Befehlskette gewartet werden, um festzustellen, daß der letzte Befehl ausgeführt worden ist. Dabei ist zu beachten, daß sich CTS im Laufe der Befehlskette mehrfach geändert hat (genau so oft, wie sich DCD geändert hat).

Nach Reset ist DCD zunächst =0.

Steckerbelegung

Stecker A

Stecker A umfaßt alle Signale, die an die Außenwelt gehen (CAN-Bus, Versorgung).

Pin	Name	I/O	Bedeutung	D-SUB
1	VCC8	I	optionaler Eingang für Spannungsversorgung des S-Links (nur nach Hardware-Umbau)	1
2	-	n.c.		-
3	CAN_L	I/O	CAN-Bus Leitung CAN-Low	2
4	GND	I/O	Ground	6
5	GND	I/O	Ground	3
6	CAN_H	I/O	CAN-Bus Leitung CAN-High	7
7-18	-	n.c.		-
19	VCC12	O	optionale Versorgungsspannung für externe Geräte (wird von Stecker B, Pin 6 durchgeschleift - nur nach Hardwareumbau)	8
20	-	n.c.		4
21	CANV5	O	5 V Versorgungsspannung (isoliert) für externe Geräte	9
22	GND	I/O	Ground	5
23,24	-	n.c		-

Die letzte Spalte gibt die Belegung eines 9-poligen D-SUB Steckers an, wie er z.B. für die Multi-COM angeboten wird. Die Belegung entspricht dem CAN-Standard (nach CiA Standard 102, Version 2.0).

Standardmäßig ist nur die Versorgung externer Geräte mit 5V vorgesehen.

Bestückungs-Optionen

- Durch Änderung der Bestückung kann zusätzlich die Versorgungsspannung von Stecker B, Pin 6 ausgegeben werden. Auf der Multi-COM liegen dort 12 V an, die auf dem S-Link jedoch nicht verwendet werden (daher können an Pin 6 von Stecker B auch andere Spannungen angelegt werden). Dadurch würde aber die galvanische Trennung aufgehoben.
- Das SL-CANi kann durch eine Änderung der Bestückung ganz oder nur sekundärseitig von Stecker A, Pin 1 mit 7 bis 35 V versorgt werden.

Die beiden beschriebenen Bestückungsänderungen stehen nur bei größeren Abnahmemengen zur Verfügung und müssen werksseitig ausgeführt werden.

Stecker B

Stecker B umfaßt alle Signale, um über eine serielle Schnittstelle mit der Steuer-CPU zu kommunizieren.

Pin	Name	I/O	Bedeutung
1	DCD	O	Statusmeldung (toggle) "Befehlspuffer frei für neuen Befehl"
2	P5	I	5 V Versorgung für das S-Link
3	RES\	I	Reset (low-active)
4	P3	n.c.	
5	DSR	O	CL=0: Zustand der CAN-Bus Empfangsleitung RX0 CL=1: EEPROM Datenausgang
6	P12	I	12 V Versorgung (nicht benötigt, siehe Bestückungs-Optionen)
7-10	-	n.c.	
11	RCV	O	Datenausgang
12	RTS	I	CL=0: Taktsignal für Befehlsempfang über DTR CL=1: EEPROM Dateneingang
13	TMT	I	Dateneingang
14	CTS	O	Acknowledge für die komplette Bearbeitung eines Befehls
15	DTR	I	CL=0: Befehlsleitung, CL=1: EEPROM Takteingang
16	RI	O	Interrupt-Ausgang (Funktionalität einstellbar, s. S. 2)
17	CTRL	I	muß =1 sein
18	CL	I	Select für EEPROM (high active)
19	CLKIO	O	Taktausgang für synchrone Übertragung
20	M12	n.c.	
21	SLX	I	muß =0 sein
22	GND	I	Ground
23	CLKIN	I	Takteingang (10 MHz)
24	GND	I	Ground

Zum genaueren Verständnis der Signale wird auf den vorhergehenden Text verwiesen. Die Spalte I/O ist aus Sicht des S-Links zu verstehen.

Technische Daten

Stromverbrauch an +5V	typ. 125 mA (LED aus, keine CAN-Kommunikation)
	max. ca. 300 mA
Stromentnahme an Stecker A, Pin 21	max. 50 mA
Trennspannung	500 V
Busabschlußwiderstand	typ. 110 Ω
Betriebstemperatur	0 bis 60 °C
Takteingang an Stecker B, Pin 23	10 MHz

Software-Beispiel

Im folgenden werden Funktionen zur Kommunikation mit dem S-Link als Auszug aus dem Treiberprogramm für die Multi-COM gezeigt. Die serielle Schnittstelle wird hier durch den Zilog Z8530 bzw. Z85230 gebildet. Die Kommunikation erfolgt synchron. Am PCLK-Eingang des SCC muß ein 7,37 MHz oder 16,5 MHz (nur Z85230) Signal anliegen. Pin 19 vom S-Link Stekker B (CLK-Ausgang) ist mit dem SCC Pin RTXC verbunden. RI liegt am SCC-Eingang SYNC. Eine Behandlung eventuell auftretender Fehler wird in diesem Beispiel nicht vorgenommen.

Der Zugriff auf die Register des SCC erfolgt in den Funktionen WriteSccReg und ReadSccReg. Dabei zu beachten, daß zwischen zwei Zugriffen auf den SCC eine Wartezeit einzuhalten ist. Außerdem darf auf alle SCC-Register außer 0 und 8 nur indirekt zugegriffen werden, d.h. die gewünschte Registernummer muß an den SCC gesendet werden (in WR00), bevor die Daten dieses Registers gelesen oder geschrieben werden können. Bei Schreibzugriffen auf den Datenpuffer WR08 muß sichergestellt sein, daß dieser leer ist. Dies kann anhand des TBE-Bits in RR00 oder des AllSent-Bits in RR01 oder durch Interruptauslösung von TBE (hier nicht implementiert) geschehen.

```
// global variables
UCHAR ucCTS, ucDTR, ucFPGA, ucLastReg, ucLastCmd;

// definitions
#define CTS_STATUS    0x20   // mask fpr CTS in RR00
#define DTRPINLO      0x80   // mask for DTR pin low in WR05
#define DTRPINHI      0x00   // mask for DTR pin high in WR05
#define SetRTSLow(UCHAR x)  WriteSccReg(WR05, 0x6A | x)
#define SetRTSHigh(UCHAR x) WriteSccReg(WR05, 0x68 | x)
```

```
/*****************************************************************************/
// init of the scc and the sl-can
void CommunicationInit(void)
{
  // Initialization of the SCC in asynchronous mode
  WriteSccReg(WR04, 0x44);   // 1 stopbit, Clock-rate = 16 x data-rate
  WriteSccReg(WR01, 0x00);   // all interrupts disabled
  WriteSccReg(WR02, 0x00);   // interrupt vector
  WriteSccReg(WR15, 0x00);   // all interrupts disabled
  WriteSccReg(WR03, 0xC0);   // 8 data bit per receive character
  WriteSccReg(WR05, 0x60);   // 8 data bit per transmit character
  WriteSccReg(WR09, 0x01);   // master interrupt disabled
  WriteSccReg(WR11, 0x50);   // use baudrate generator for tmt and rcv clock
  WriteSccReg(WR12, 0x00);   // baudrate generator = 115,2 kBit/s
  WriteSccReg(WR13, 0x00);   // baudrate generator = 115,2 kBit/s
  WriteSccReg(WR14, 0x82);   // DPLL-source = baudrate generator
  // enable the SCC functions
  WriteSccReg(WR03, 0xC1);   // enable receiver
  WriteSccReg(WR05, 0xEA);   // enable transmiter, DTR and RTS pins = 0;
  WriteSccReg(WR00, 0x80);   // reset transmiter crc
  WriteSccReg(WR14, 0x03);   // enable baudrate generator, source = PCLK
  // Reset of the S-Link (DTR is 0)
  SetRTSHigh(DTRPINLO);
  SetRTSLow(DTRPINLO);
  ucCTS = STATUS_CTS;        // init CTS variable SCCs CTS input is inverted
  WaitForCTS();
  // send 2 bytes to fpga for baudrate recognition
  WriteSCCReg(WR08, 0x80);
  WriteSCCReg(WR08, 0x55);
  WaitForCTS();
  WriteSCCReg(WR08, 0x0B);   // use synchronous 1,25 MHz communication, R on
  WaitForCTS();
  // disable the SCC functions for reinit of the SCC
  WriteSccReg(WR03, 0xC0);   // disable receiver
  WriteSccReg(WR05, 0xE2);   // disable transmiter, DTR and RTS pins = 0;
  WriteSccReg(WR14, 0x00);   // disable baudrate generator
  // change SCC settings to synchronous mode
  WriteSccReg(WR04, 0x04);   // 1 stopbit, Clock-rate = data-rate
  WriteSccReg(WR11, 0x00);   // use RTXC input for tmt and rcv clock
  WriteSccReg(WR14, 0xA0);   // DPLL-source = RTXC input
  // enable the SCC functions
  WriteSccReg(WR03, 0xC1);   // enable receiver
  WriteSccReg(WR05, 0x6A);   // enable transmiter, RTS pin = 0;
  WriteSccReg(WR00, 0x80);   // reset transmiter crc
  WriteSccReg(WR14, 0x01);   // enable baudrate generator
  // send command to read out the fpga version
  SetRTSHigh(DTRPINHI);
  SetRTSHigh(DTRPINLO);      // DTR to 0
  SetRTSLow(DTRPINLO);
  WaitForCTS();
  ucFPGA    = ReadSccReg(RR08) & 0x0F;
  ucDTR     = 0;
  ucLastReg = 0xFF;          // force first can-access with new address
  ucLastCmd = RESET_CMD;     // force first can-access with new command
}

/*****************************************************************************/
// read a register of the CAN controller
UCHAR ReadCanReg(UCHAR ucRegNo)
{
  if (ucLastCmd == READ_CMD) {
    WriteSccReg(WR08, ucRegNo);   // write register address to sl-can
    ucLastReg = ucRegNo;
  }
```

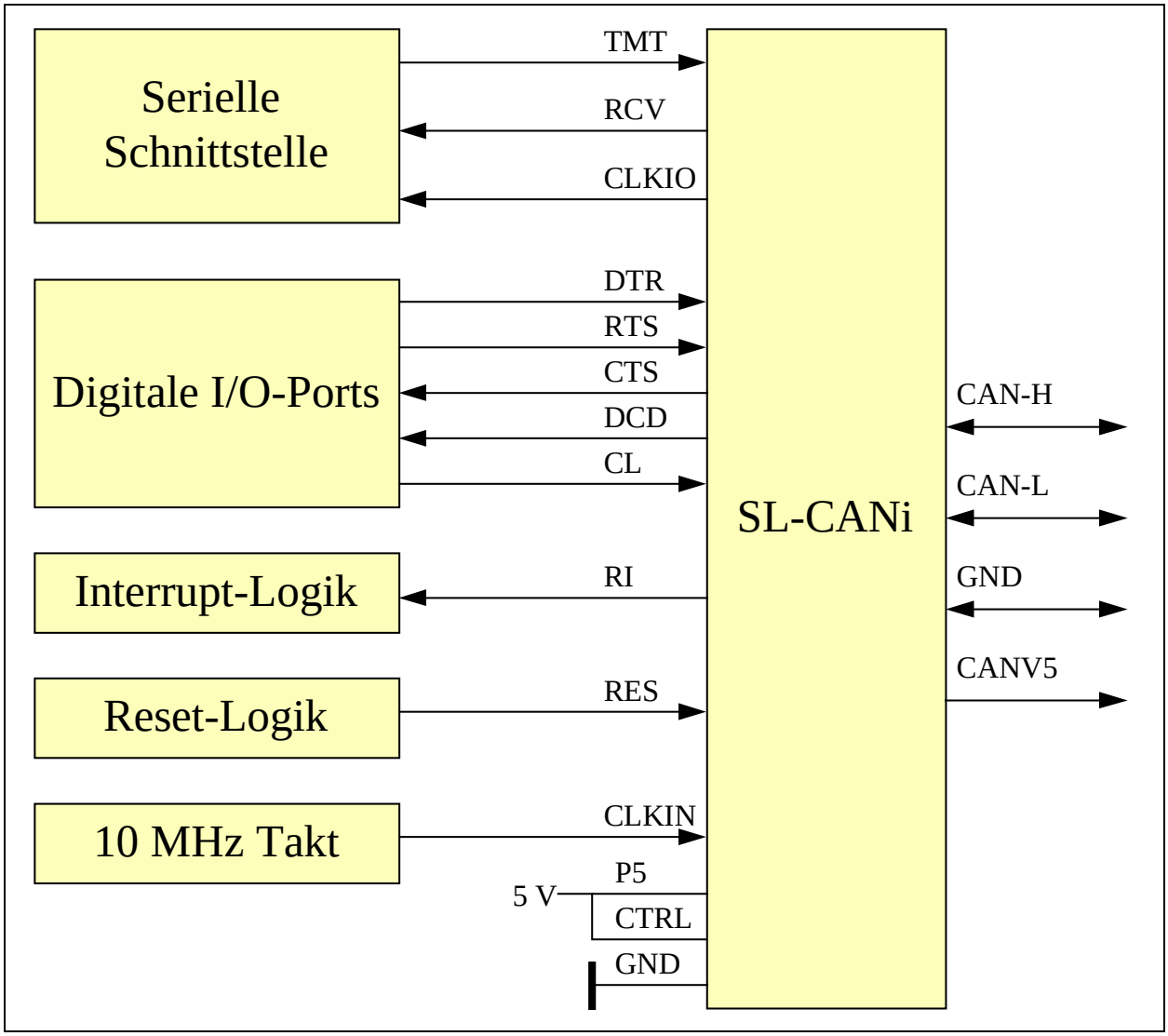
```
  else {
    if (ucLastReg == ucRegNo) {   // use <read next reg> command
      if (ucDTR == 0) SetRTSLow(DTRPINHI); // set DTR=1 if it is 0
      SetRTSHigh(DTRPINHI);
      SetRTSHigh(DTRPINLO);
      SetRTSLow(DTRPINLO);
      ucDTR = 0;
    }
    else {                        // use <read new reg> command
      if (ucDTR == 1) SetRTSLow(DTRPINLO); // set DTR=0 if it is 1
      SetRTSHigh(DTRPINLO);
      SetRTSHigh(DTRPINHI);
      SetRTSLow(DTRPINHI);
      ucDTR = 1;
      WriteSccReg(WR08, ucRegNo); // write register address to sl-can
      ucLastReg = ucRegNo;
    }
    ucLastCmd = READ_CMD;
  }
  ucLastReg++;
  WaitForCTS();
  return(ReadSccReg(RR08));
}

/*****************************************************************************/
// write a register of the CAN controller
void WriteCanReg(UCHAR ucRegNo, UCHAR ucData)
{
  if ((ucLastReg != ucRegNo) || (ucLastCmd != WRITE_CMD)){
    if (ucDTR == 0) SetRTSLow(DTRPINHI); // set DTR=1 if it is 0
    SetRTSHigh(DTRPINHI);
    SetRTSLow(DTRPINHI);
    ucDTR = 1;
    WriteSccReg(WR08, ucRegNo);        // write register address to sl-can
    WaitTBE();
    ucLastReg = ucRegNo;
  }
  WriteSccReg(WR08, ucData);           // write data into register
  ucLastCmd = WRITE_CMD;
  ucLastReg++;
  WaitForCTS();
}

/*****************************************************************************/
// function to check CTS-toggeling, the ucCTS variable will be updated
USHORT WaitForCTS(void)
{
  UCHAR  ucCTSStatus;
  USHORT usTimeout = CTS_TIMEOUT;

  do    ucCTSStatus = ReadSccReg(RR00) & STATUS_CTS;
  while ((ucCTSStatus == ucCTS) && (--usTimeout != 0));
  if (usTimeout == 0) return(1);  // communication timeout
  ucCTS = ucCTSStatus;
  return(0);
}
```

Anhang: Anschluß des SL-CANi an ein Host-System


Serielle Schnittstelle
Digitale I/O-Ports
Interrupt-Logik
Reset-Logik
10 MHz Takt
SL-CANi
TMT
RCV
CLKIO
DTR
RTS
CTS
DCD
CL
RI
RES
CLKIN
P5
5 V
CTRL
GND
CAN-H
CAN-L
GND
CANV5