5 Electrical Characteristics

Absolute Maximum Ratings

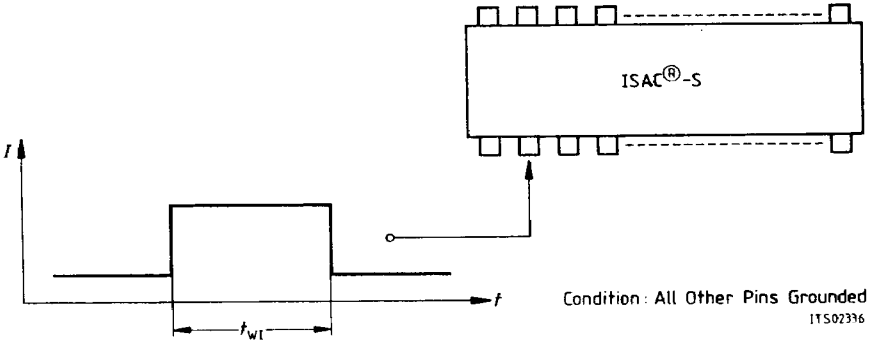
Parameter	Symbol	Limit Values	Unit
Voltage on any pin with respect to ground	V_S	– 0.4 to V_{DD} + 0.4	V
Ambient temperature under bias	T_A	0 to 70	°C
Storage temperature	T_{stg}	– 65 to 125	°C
Maximum voltage on V_{DD}	V_{DD}	6	V

Note: Stresses above those listed here may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Line Overload Protection

The maximum input current (under overvoltage conditions) is given as a function of the width of a rectangular input current pulse (**figure 73**).

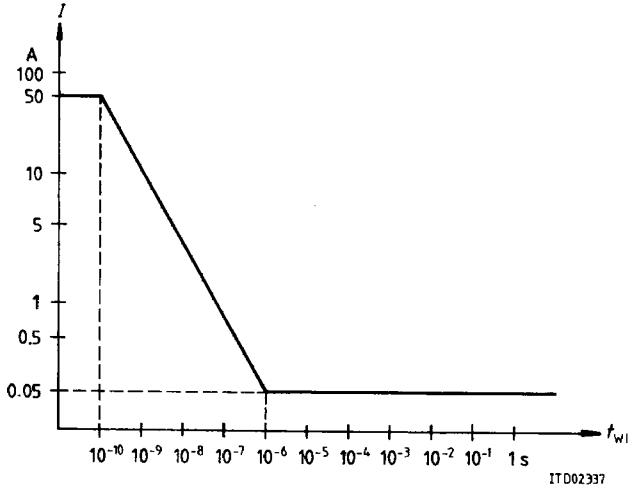
Figure 73
Test Condition for Maximum Input Current



Transmitter Input Current

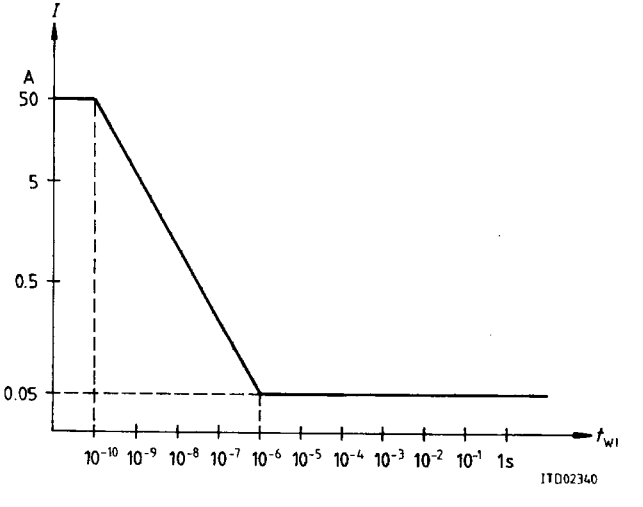
The destruction limits for negative input signals are given in **figure 74**. $R_i \geq 2\ \Omega$.

Figure 74



The destruction limits for positive input signals are given in **figure 75**. $R_i \geq 200\ \Omega$.

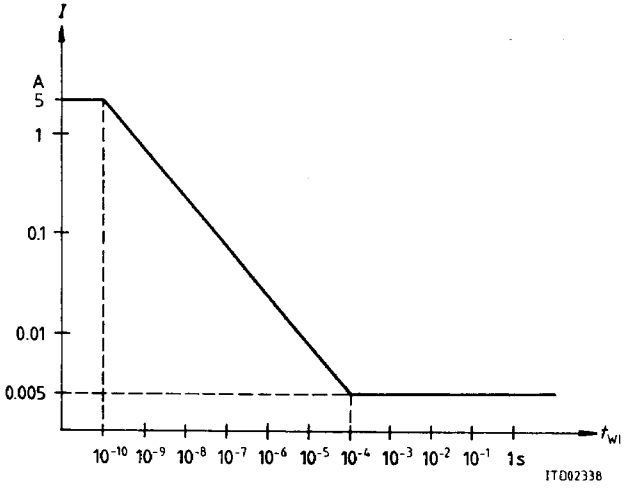
Figure 75



Receiver Input Current

The destruction limits are given in **figure 76**. $R_i \geq 300\ \Omega$.

Figure 76



DC Characteristics

T_A = 0 to 70 °C; V_{DD} = 5 V ± 5 %, V_{SSA} = 0 V, V_{SSD} = 0 V

Parameter		Symbol	Limit Values		Unit	Test Condition		Remarks
			min.	max.				
L-input voltage		V_{IL}	– 0.4	0.8	V			
H-input voltage		V_{IH}	2.0	V_{DD} + 0.4	V			
L-output voltage L-output voltage (IDPO)		V_{OL} V_{OL1}		0.45 0.45	V V	I_{OL} = 2 mA I_{OL} = 7 mA		All pins exc.t SX1,2, SR1,2
H-output voltage H-output voltage		V_{OH} V_{OH}	2.4 V_{DD} – 0.5		V V	I_{OH} = – 400 µA I_{OH} = – 100 µA		
Power supply current	power down	I_{CC}		1.5	mA		V_{DD} = 5 V Inputs at V_{SS} / V_{DD} No output loads	
	oper-ational			15	mA	DCL = 512 kHz		
				17	mA	DCL = 1536 kHz		
				22	mA	DCL = 4096 kHz		
Input leakage current Output leakage current		I_{LI} I_{LO}		10 10	µA µA	0 V < V_{IN} < V_{DD} to 0 V 0 V < V_{OUT} < V_{DD} to 0 V		All pins except SX1,2, SR1,2
Absolute value of output pulse amplitude (VSX2 – VSX1)		V_X	2.03 2.10	2.31 2.39	V V	R_L = 50 Ω [1] R_L = 400 Ω [1]		
Transmitter output current		I_X	7.5	13.4	mA	R_L = 5.6 Ω [1]		SX1,2
Transmitter output impedance		R_X	10 0		kΩ Ω	inactive or during binary one during binary zero R_L = 50 Ω		
Receiver output voltage		V_{SR1}	2.35	2.6	V	I_O < 5 µA		
Receiver threshold voltage $V_{SR2} - V_{SR1}$		V_{TR}	225	375	mV	Dependent on peak level		SR1,2

Note: [1] Due to the transformer, the load resistance seen by the circuit is four times R_L.

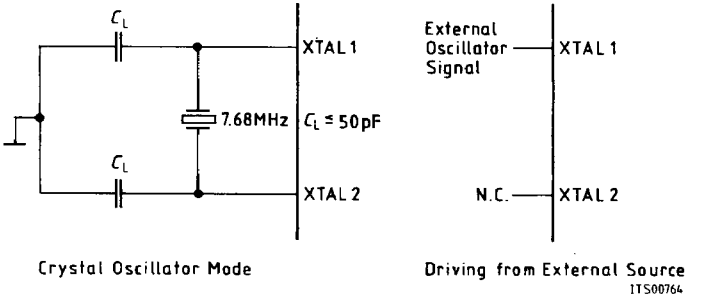
Capacitances

T_A = 25 °C, V_{DD} = 5 V ± 5 %, V_{SSA} = 0 V, V_{SSD} = 0 V, f_c = 1 MHz, unmeasured pins grounded.

Parameter	Symbol	Limit Values		Unit	Remarks
		min.	max.		
Input capacitance I/O capacitance	C_{IN} $C_{I/O}$		7 7	pF pF	All pins except SR1,2, XTAL1,2
Output capacitance against V_{SSA}	C_{OUT}		10	pF	SX1,2
Input capacitance	C_{IN}		7	pF	SR1,2
Load capacitance	C_L		50	pF	XTAL1,2

Recommended Oscillator Circuits

Figure 77
Oscillator Circuits



Crystal Specification

Parameter	Symbol	Limit Values	Unit
Frequency	f	7.680	MHz
Frequency calibration tolerance		max. 100	ppm
Load capacitance	C_L	max. 50	pF
Oscillator mode		fundamental	

Note: The load capacitance C_L depends on the recommendation of the crystal specification. Typical values for C_L are 18 ... 22 pF.

XTAL1 Clock Characteristics (external oscillator input)

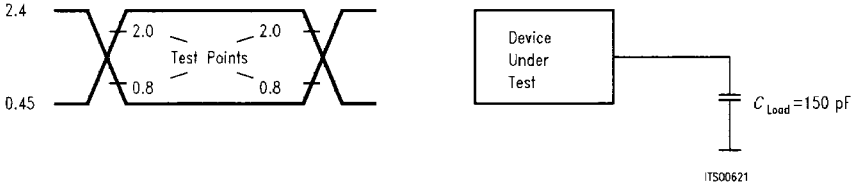
Parameter	Limit Values	
	min.	max.
Duty cycle	1:2	2:1

AC Characteristics

T_A = 0 to 70 °C, V_{DD} = 5 V ± 5%

Inputs are driven to 2.4 V for a logical "1" and to 0.4 V for a logical "0". Timing measurements are made at 2.0 V for a logical "1" and 0.8 V for a logical "0". The AC testing input/output waveforms are shown in **figure 78.**

Figure 78
Input/Output Waveform for AC Tests



Microprocessor Interface Timing

Siemens/Intel Bus Mode

Figure 79
Microprocessor Read Cycle

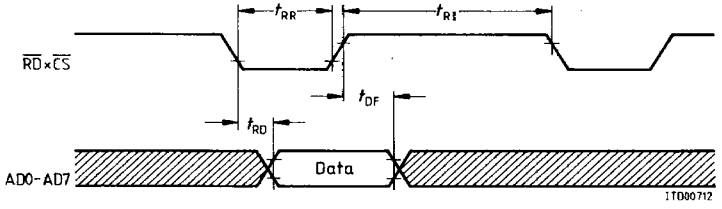


Figure 80
Microprocessor Write Cycle

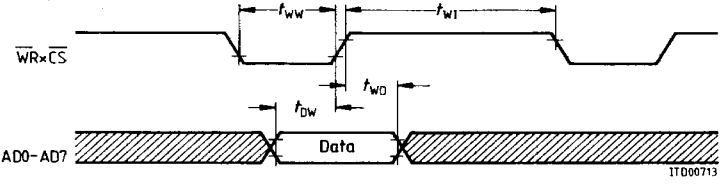


Figure 81
Multiplexed Address Timing

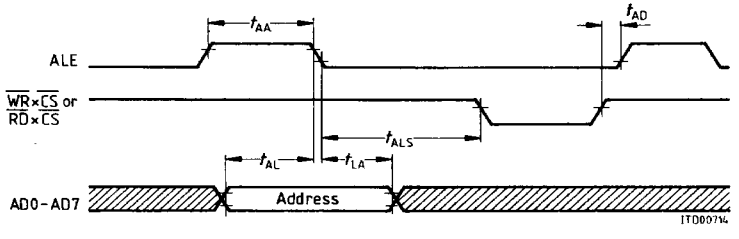
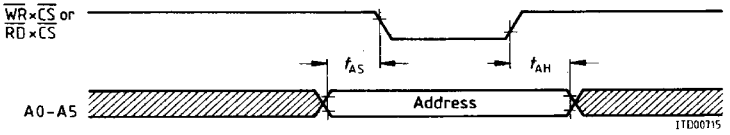


Figure 82
Non-Multiplexed Address Timing



Motorola Bus Mode

Figure 83
Microprocessor Read Timing

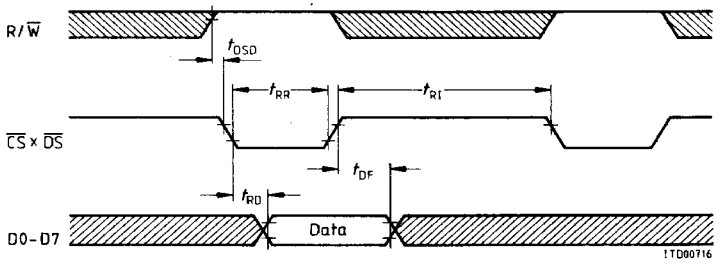


Figure 84
Microprocessor Write Cycle

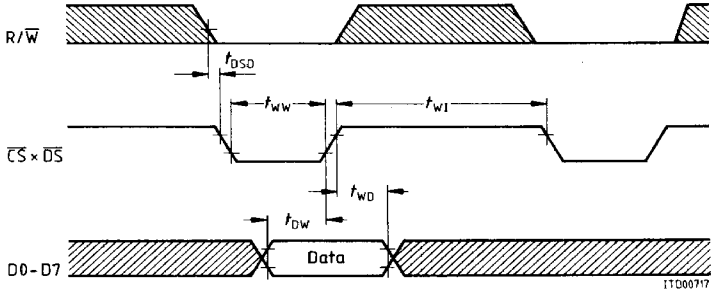
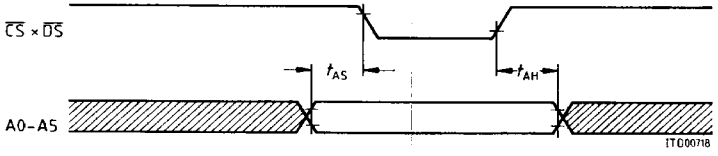


Figure 85
Non-Multiplexed Address Timing



Microprocessor Interface Timing

Parameter	Symbol	Limit Values		Unit
		min.	max.	
ALE pulse width	t_{AA}	50		ns
Address setup time to ALE	t_{AL}	15		ns
Address hold time from ALE	t_{LA}	10		ns
Address latch setup time to $\overline{WR}$, $\overline{RD}$	t_{ALS}	0		ns
Address setup time	t_{AS}	25		ns
Address hold time	t_{AH}	10		ns
ALE guard time	t_{AD}	15		ns
$\overline{DS}$ delay after R/$\overline{W}$ setup	t_{DSD}	0		ns
$\overline{RD}$ pulse width	t_{RR}	110		ns
Data output delay from $\overline{RD}$	t_{RD}		110	ns
Data float from $\overline{RD}$	t_{DF}		25	ns
$\overline{RD}$ control interval	t_{RI}	70		ns
$\overline{WR}$ pulse width	t_{WW}	60		ns
Data setup time to $\overline{WR} \times \overline{CS}$	t_{DW}	35		ns
Data hold time from $\overline{WR} \times \overline{CS}$	t_{WD}	10		ns
$\overline{WR}$ control interval	t_{WI}	70		ns

Serial Interface Timing

Figure 86
IOM® Timing (TE mode)

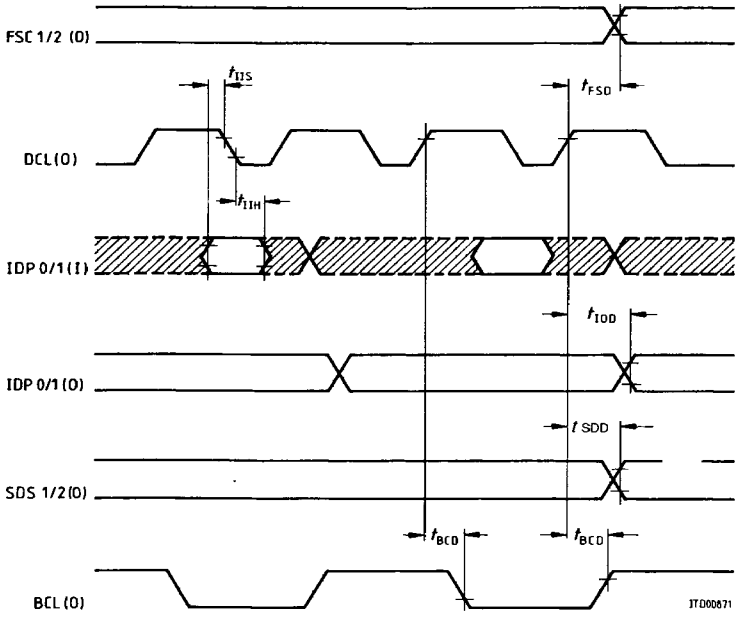
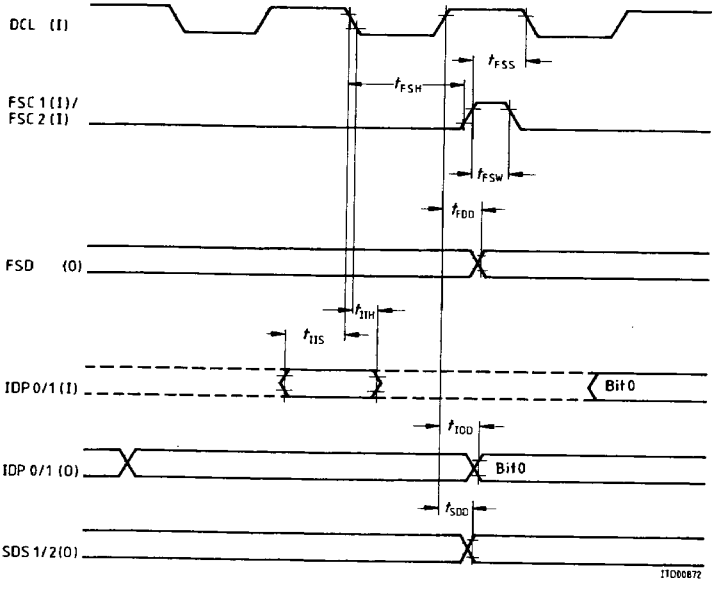


Figure 87

IOM® Timing (LT-S, LT-T, NT mode)

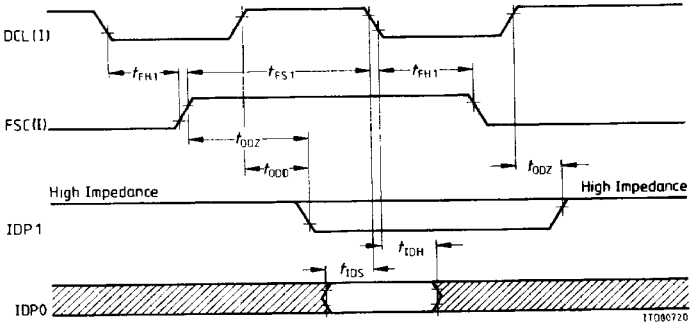


IOM® Timing

Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
IOM output data delay	t_{IOD}	20 20	140 100	ns ns	IOM-1 IOM-2
IOM input data setup	t_{IIS}	$4 + t_{WH}$ 20		ns ns	IOM-1 IOM-2
IOM input data hold	t_{IIH}	20		ns	
FSC1/2 strobe delay	t_{FSD}	– 20	20	ns	
Strobe signal delay	t_{SDD}		120	ns	
Bit clock delay	t_{BCD}	– 20	20	ns	
Frame sync setup	t_{FSS}	50		ns	
Frame sync hold	t_{FSH}	30		ns	
Frame sync width	t_{FSW}	40		ns	
FSD delay	t_{FDD}	20	140	ns	

HDLC Mode (ADF2: IMS = 0, ADF1: TEM = 1, MODE: DIM2 – 0 = 101 – 111)

Figure 88
FSC1 (strobe) Characteristics

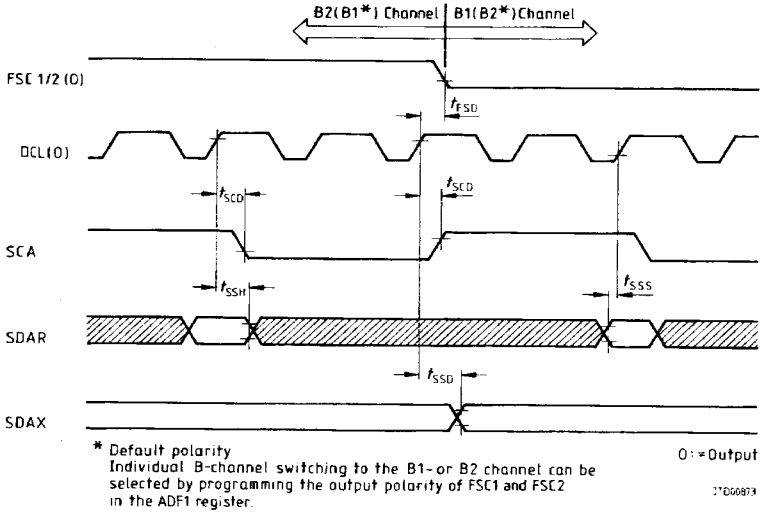


HDLC Mode Timing

Parameter	Symbol	Limit Values		Unit
		min.	max.	
FSC1 set-up time	t_{FS1}	100		ns
FSC1 hold time	t_{FH1}	30		ns
Output data from high impedance to active	t_{OZD}		80	ns
Output data from active to high impedance	t_{ODZ}		40	ns
Output data delay from DCL	t_{ODD}	20	100	ns
Input data setup	t_{IS}	10		ns
Input data hold	t_{DH}	30		ns

Serial Port A (SSI) Timing

Figure 89
SSI Timing (TE, timing mode 0)



Serial Port A (SSI) Timing

Parameter	Symbol	Limit Values		Unit
		min.	max.	
SCA clock delay	t_{SCD}	20	140	ns
SSI data delay	t_{SSD}	20	140	ns
SSI data setup	t_{SSS}	40		ns
SSI data hold	t_{SSH}	20		ns
FSC1/2 strobe delay	t_{FSD}	– 20	20	ns

SLD Timing

Figure 90
SLD Timing (TE mode)

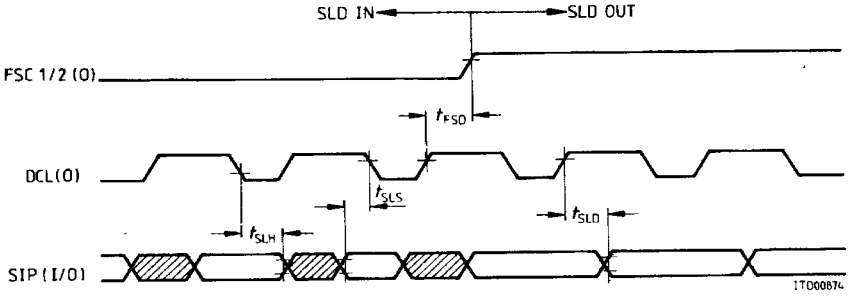
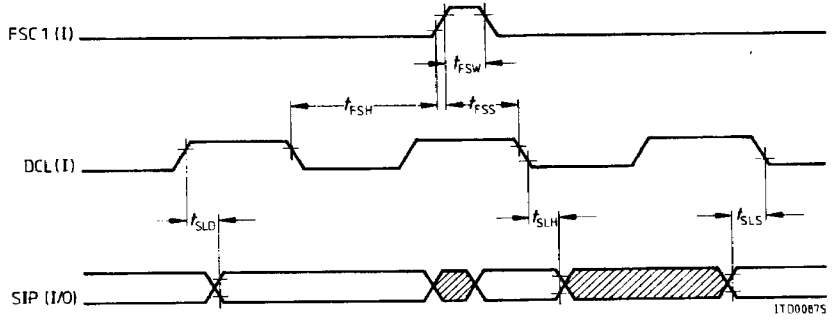


Figure 91
SLD Timing (LT-S / LT-T mode)



SLD Timing

Parameter	Symbol	Limit Values		Unit
		min.	max.	
SLD data delay	t_{SLD}	20	140	ns
SLD data setup	t_{SLS}	30		ns
SLD data hold	t_{SLH}	30		ns
FSC1/2 strobe delay	t_{FSD}	– 20	20	ns
Frame sync setup	t_{FSS}	50		ns
Frame sync hold	t_{FSH}	30		ns
Frame sync width	t_{FSW}	40		ns

Clock Timing

The clocks in the different operating modes are summarized in **tables 23 – 25**, with the respective duty ratios.

Table 23
ISAC®-S Clock Signals (IOM®-1 mode)

Application	M1	M0	DCLK	FSC1/2	CP	X1
TE	0	0	o:512 kHz* 1:2	o:8 kHz* 1:1	o:1536 kHz* 3:2	o:3840 kHz 1:1
LT-T	0	1	i:512 kHz	i:8 kHz	–	o:7680 kHz 1:1
LT-S	1	0	i:512 kHz	i:8 kHz	o:512 kHz* 1:2	–
NT	1	1	i:512 kHz	i:8 kHz	–	–

Table 24
ISAC®-S Clock Signals (IOM®-2 mode)

Application	M1	M0	DCL	FSC1	FSC2	CP/BCL	X1	SDS1/2
TE	0	0	o:1536 kHz* 3:2	o:8 kHz* 1:2		o:768 kHz* 1:1	–	o:8 kHz 1:11 2:10
LT-T	0	1	i:4096 kHz	i:8 kHz	i:8 kHz	–	o:7680 kHz 1:1	o:8 kHz 1:11 2:10
LT-S	1	0	i:4096 kHz	i:8 kHz	i:8 kHz	o:512 kHz* 1:2	–	o:8 kHz 1:11 2:10
NT	1	1	i:512 kHz	i:8 kHz	i:8 kHz	–	–	o:8 kHz 1:11 2:10

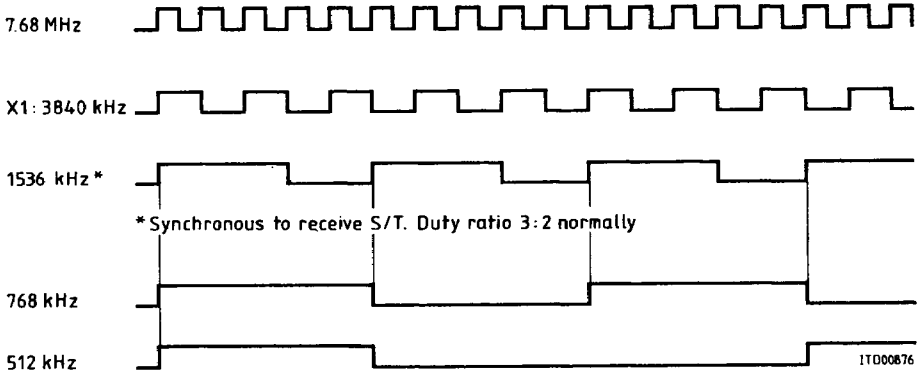
*) Synchronous to receive "S" line

The 1536-kHz clock (TE mode) and the 512-kHz clock (LT-T mode) are phase-locked to the receive S signal, and derived using the internal DPLL and the 7.68 MHz ± 100 ppm crystal.

A phase tracking with respect to "S" is performed once in 250 µs. As a consequence of this DPLL tracking, the "high" state of the 1536-kHz clock may be either reduced or extended by one 7.68-MHz period (duty ratio 2:2 or 4:2 instead of 3:2) once every 250 µs. Since the other signals are derived from this clock (TE mode), the "high" or "low" states may likewise be reduced or extended by the same amount once every 250 µs.

The phase relationships of the clocks are shown in **figure 92**.

Figure 92
Phase Relationships of ISAC®-S Clock Signals



The timing relationships between the clocks are specified in **figure 89** and **table 26**.

Figure 93
Timing Relationships between ISAC®-S Clock Signals

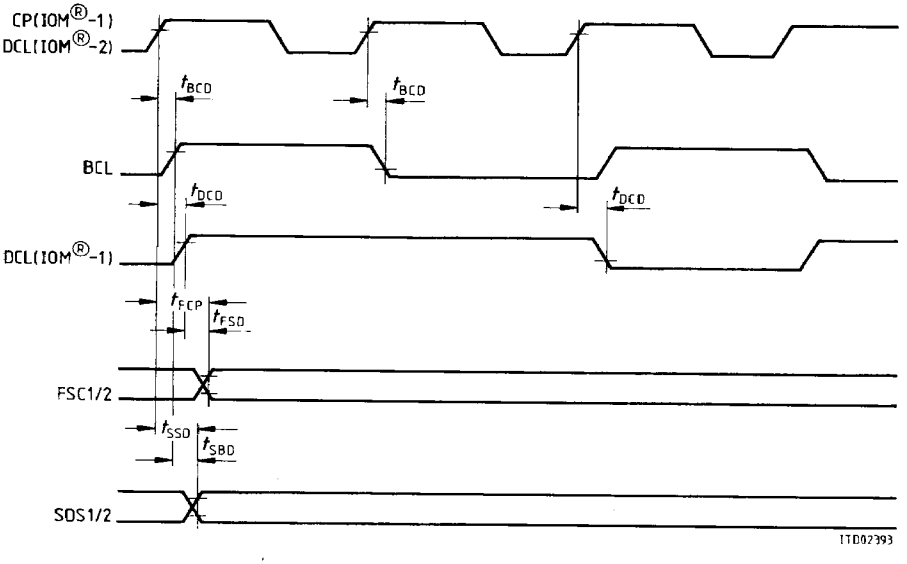


Table 25

Parameter	Symbol	Limit Values		Unit	Conditions
		min.	max.		
Bit clock delay	t_{BCD}	– 20	20	ns	IOM-2
SDS1/2 delay from DCL	t_{SDD}		120	ns	IOM-2
SDS1/2 delay from BCL	t_{SBD}		120	ns	IOM-2
DCL delay from CP	t_{DCD}	0	50	ns	IOM-1
FSC1/2 delay from CP	t_{FCP}	0	50	ns	IOM-1
FSC1/2 delay from DCL	t_{FSD}	– 20	20	ns	IOM-1

Tables 27 to **31** give the timing characteristics of the clocks.

Figure 94
Definition of Clock Period and Width

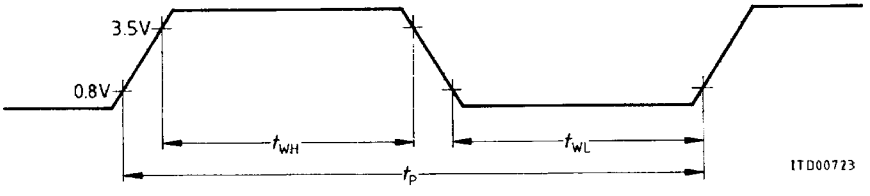


Table 26
DCL Clock Characteristics (IOM®-1)

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	typ.	max.		
(TE) 512 kHz	t_{PO}	1822	1953	2084	ns	osc ± 100 ppm
(TE) 512 kHz 1:2	t_{WHO}	470	651	832	ns	osc ± 100 ppm
(TE) 512 kHz 1:2	t_{WLO}	1121	1302	1483	ns	osc ± 100 ppm
(NT, LT-S, LT-T)	t_{PI}	1853		2053	ns	
(NT, LT-S, LT-T)	t_{WHI}	200			ns	
(NT, LT-S, LT-T)	t_{WLI}	200			ns	

Table 27
DCL Clock Characteristics (IOM®-2)

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	typ.	max.		
(TE) 1536 kHz	t_{PO}	520	651	782	ns	osc ± 100 ppm
	t_{WHO}	240	391	541	ns	osc ± 100 ppm
	t_{WLO}	240	260	281	ns	osc ± 100 ppm
(LT-S, LT-T) 4096 kHz	t_{PI}	240	244		ns	
	t_{WHI}	100			ns	
	t_{WLI}	100			ns	

Note: For NT characteristics, see IOM-1 case.

Table 28
CP Clock Characteristics (IOM®-1 TE mode)

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	typ.	max.		
(TE) 1536 kHz	t_{PO}	520	651	782	ns	osc ± 100 ppm
	t_{WHO}	240	391	541	ns	osc ± 100 ppm
	t_{WLO}	240	260	281	ns	osc ± 100 ppm

Table 29
CP Clock Characteristics (LT-T mode)

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	typ.	max.		
(LT-T) 512 kHz	t_{PO}	1822	1953	2084	ns	osc ± 100 ppm
	t_{WHO}	470	651	832	ns	osc ± 100 ppm
	t_{WLO}	1121	1302	1483	ns	osc ± 100 ppm

Table 30
X1 Clock Characteristics (TE mode)

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	typ.	max.		
(TE) 3840 kHz	t_{PO}	– 100 ppm	260	100 ppm	ns	osc ± 100 ppm
	t_{WHO}	120	130	140	ns	osc ± 100 ppm
	t_{WLO}	120	130	140	ns	osc ± 100 ppm

Table 31
X1 Clock Characteristics (LT-S mode)

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	typ.	max.		
(LT-S) 7680 kHz	t_{PO}	– 100 ppm	130.21	100 ppm	ns	osc ± 100 ppm
	t_{WHO}		65		ns	osc ± 100 ppm
	t_{WLO}		65		ns	osc ± 100 ppm

Jitter

In TE mode, the timing extraction jitter of the ISAC-S conforms to CCITT Recommendation I.430 (– 7% to + 7% of the S-interface bit period).

In the NT and LT-S applications, the clock input DCL is used as reference clock to provide the 192-kHz clock for the S-line interface. In the case of a plesiochronous 7.68-MHz clock generated by an oscillator, the clock DCL should have a jitter less than 100 ns peak-to-peak. (In the case of a zero input jitter on DCL the ISAC-S generates at most 130 ns "self-jitter" on the S interface.)

In the case of a synchronous *) 7.68-MHz clock (input XTAL1), the ISAC-S transfers the input jitter of XTAL1, DCL and FSC1 to the S interface. The maximum jitter of the NT/LT-S output is limited to 260 ns peak-to-peak (CCITT I.430).

Description of the Transmit PLL (XPLL) of the ISAC®-S

Function of the XPLL

The XPLL generates a 1.536-MHz clock synchronized to the DCL 512-kHz clock by modification of the counter's divider ratio. The 1.536-MHz clock is then divided to 192 kHz and 512 kHz. The 512 kHz is used as the looped back clock and compared to the 512-kHz DCL in the phase detector. A four bit up/down counter integrates the phase information to prevent tracking steps in presence of high frequency input jitter **(see figure 95)**.

Jitter considerations in case of a synchronous 7.68-MHz clock

After the XPLL has locked once, no more tracking steps are performed because there is a fixed divider ratio of 15 between 7.68 MHz and DCL. Therefore the input jitter at DCL and 7.68 MHz is transferred transparently to the S/T interface (192 kHz).

Jitter considerations in case of a plesiochronous 7.68-MHz clock (crystal)

Each tracking step of the XPLL produces an output jitter of 130 ns pp. In case of non-zero input jitter at DCL, this input jitter is increased by 130 ns pp. However, if the input jitter frequency is high enough (in the range of 25 kHz and higher) the four bit up/dn counter works as a loop filter and thus the XPLL attenuates the input jitter to zero.

*) fixed divider ratio between XTAL1 and DCL

That means that the output jitter will not exceed 130 ns pp. In the intermediate range of jitter frequency, the degree of jitter attenuation lies between zero and the maximum (**see figure 96**).

Figure 95
Block Diagram of XPLL

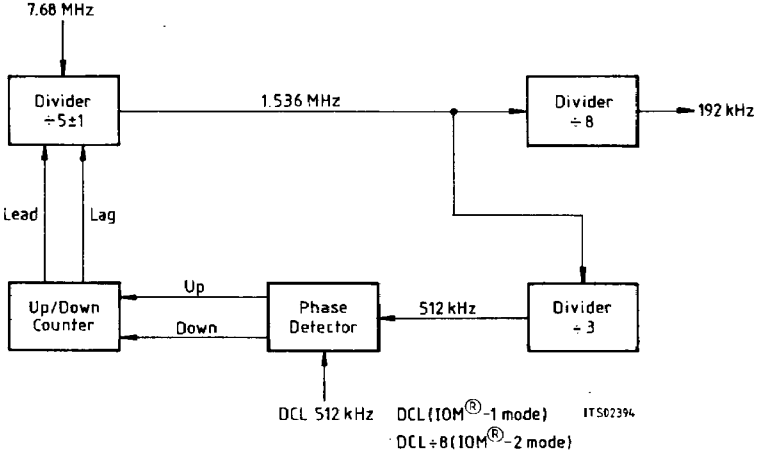
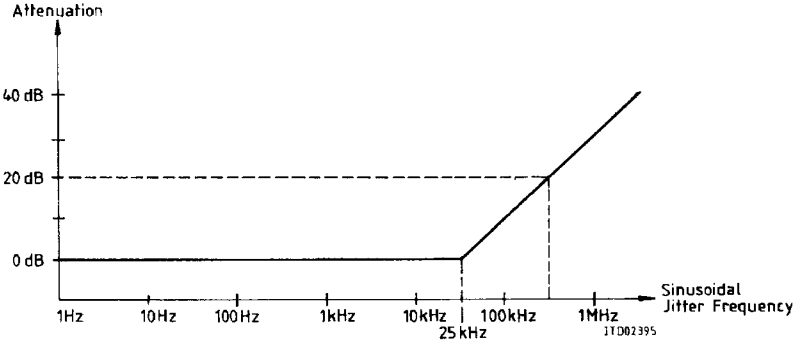


Figure 96
Jitter Transfer Curve of XPLL



Description of the receive PLL (RPLL) of the ISAC-S

The receive PLL performs phase tracking each 250 µs after detecting the phase between the F/L transition of the receive signal and the recovered clock. Phase adjustment is done by adding or subtracting 130 ns to or form a 1.536-MHz clock cycle. The 1.536-MHz clock is than used to generate any other clock synchronized to the line.

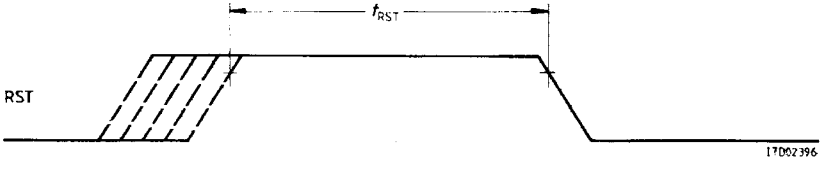
During (re)synchronization an internal reset condition may effect the 1.536-MHz and 512-kHz clocks to have high or low times as short as 130 ns. After the S/T interface frame has achieved the synchronized state (after three consecutive valid pairs of code violations) the FSC output in TE mode is set to a specific phase relationship, thus causing once an irregular FSC timing.

Reset

Table 32
Reset Signal Characteristics

Parameter	Symbol	Limit Values	Unit	Test Condition
		min.		
Length of active high state	t_{RST}	4	ms	Power on/Power Down to Power Up (Standby)
		2* DCL clock cycles		During Power Up (Standby)

Figure 97



6 ISAC®-S Low Level Controller

The following paragraphs outline the functionality and structure of a software driver example for the ISAC-S. This example is based on the Low Level Controllers (LLC's) of the Siemens ISDN PC mainboard firmware (SIPB MF) which are available in C source code. This ISAC-S software driver will be also referred to as LLC or ISAC-S LLC.

It should be noted that the ISAC-S LLC does not access the complete palette of device functions but rather a subset of them. For example it is only operational in the IOM-2 TE configuration and not all message transfer modes are supported. Please refer to the paragraph titled 'Architecture and Functions' for a more detailed description.

The ISAC-S LLC presented here has been successfully tested in the Siemens ISDN PC development system. Correct operation with a higher layer software has been verified by using the Siemens ISDN Software Development and Evaluation System (SIDES) and the Siemens ISDN Operational Software (IOS).

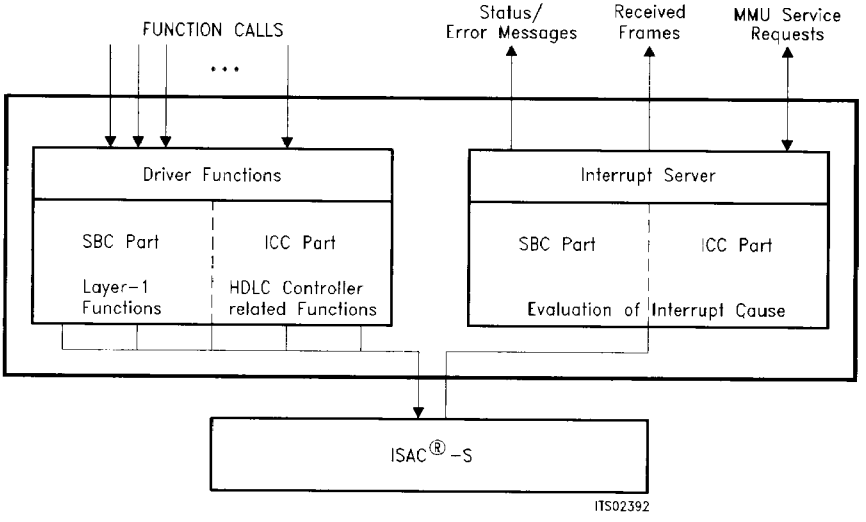
6.1 Architecture and Functions

The ISAC-S LLC may be divided into two major parts, one for layer-1 control, the 'SBC part', and one for directing the HDLC controller operations, the 'ICC part'. The naming conventions 'SBC part' and 'ICC part' have been introduced for two reasons: The first is that the ISAC-S may be viewed as the one-chip integration of the Siemens ISDN Communications Controller, PEB 2070 ICC, and the S-Bus Interface Circuit, PEB 2080 SBC. The second is that the SIPB mainboard firmware, the basis for this example software, actually uses the same code to control either an ISAC-S or an ICC-SBC combination.

The ISAC-S LLC consists of **driver functions** and **interrupt server**. The driver functions are implemented as a set of C functions which are responsible for interpreting hardware related commands from the higher layers and carrying out the appropriate actions at the hardware level. Driven by hardware interrupts, the interrupt server analyses the hardware event and informs the higher software layers of that event.

It should be noted that this implementation has attempted to remove as many protocol specific functions as possible from the LLC and to place them instead in the higher layer protocol itself. This has the advantage of making the LLC more general and less likely to be in need of re-programming for different protocols.

Figure 98
LLC Architecture



The ISAC-S LLC supports following standard functions:

- Initialization of the SBC (layer 1) part.
- Activation of layer 1.
  HDLC controller initialization.
  The following HDLC controller message transfer modes are supported:
  auto-mode: full two byte address compare, LAPD support.
  non-auto-mode: full two byte address compare.
  transparent mode 3: high byte address compare; called 'TRANSPARENT' mode in the LLC.
  transparent mode 2: no address compare; called 'EXTENDED TRANSPARENT' mode in the LLC.
  HDLC framing with two byte address field is assumed.

- HDLC frame transmission.
- Programming of TEI and SAPI values.
- HDLC transceiver control.
- Interrupt handling.
- Local test loop switching.

The LLC assumes that the ISAC-S is operating in an IOM-2 TE configuration.

In addition to the ISAC-S standard functions supporting the ISDN basic access, the ISAC-S contains optional, terminal specific functions. These terminal specific functions (watchdog and external awake) are not supported by this LLC.

6.2 Summary of LLC Functions

6.2.1 Layer 1 Related Functions

Mnemonic	Purpose
ActL1_SBC	Layer-1 activation.
ArlL1_SBC	Activation of a loop 3.
EnaClk_SBC	Enable clocking in power down mode.
InitL1_SBC	Layer-1 initialization and reset.
ResL1_SBC	Layer-1 reset.
IntL1_SBC	Handling of CISQ interrupts.

The layer 1 related functions call DECODE_L1_STATUS to report a L1 status change to a higher layer software.

6.2.2 HDLC Controller Related Functions

Mnemonic	Purpose
InitLay2_ICC	HDLC controller initialization.
Loop_ICC	Testloop activation at the serial outputs of the IOM interface.
ResetHDLC_ICC	HDLC transceiver reset.
RecReady_ICC	Setting the HDLC receiver ready or not ready.
SendFrame_ICC	HDLC frame transmission.
StoreSAPI_ICC	SAPI programming.
StoreTEI_ICC	TEI programming.
Int_ICC	Handling of XPR, RSC, TIN and EXI interrupts.
Rx_ICC	Handling of RPF and RME interrupts.

6.2.3 External Functions

The LLC program listing shows some references to external functions (indicated by an 'IMPORT' declaration). These functions are used by the LLC but are not part of it. These external functions must be provided by the operating system or a higher layer protocol software.

MMU_req ()

By calling MMU_req the ISAC-S LLC requests memory for the temporary storage of a received data frame. The memory management unit (MMU) of the operating system has to provide a memory buffer of the required size (max. 260 bytes).

MMU_free ()

MMU_free is the counterpart to MMU_req. The operating system can release a previously allocated memory buffer.

STRING_IN () and STRING_OUT ()

STRING_IN and STRING_OUT are assembler written functions for fast input and output of data frames from/to the ISAC-S FIFO.

ENTERNOINT () and LEAVENOINT ()

ENTERNOINT and LEAVENOINT are called to disable and enable all system interrupts in time critical sections.

Decode_S_Frame_BASIC ()

Decode_S_Frame_BASIC is called by the LLC interrupt server to transfer a received HDLC S frame to a higher layer protocol software.

Following information is passed to Decode_S_Frame_BASIC:

'pei': 1 byte value identifying the performed address recognition. The bits 0, 1 and 2 of 'pei' represent the bits TA, SA0 and SA1 of the ISAC-S' RSTA register.

'sapi': 1 byte value representing the received HDLC SAPI address byte. Bit 1 of 'sapi' is the C/R bit value (RSTA:CR). The most significant 6 bits of 'sapi' are 0 in auto-mode, non-auto-mode and transparent mode.

'tei': 1 byte value representing the received HDLC TEI address byte. 'tei' is 0 in auto-mode and non-auto-mode.

'ctrl': 2 byte value representing the contents of the received HDLC control field.

'frame_status': 1 byte value

= 0 × 00: frame is valid.
= 0 × 80: frame is mutilated (last byte of two byte control field missing).
= 0 × 82: frame is too long. S-frame with I-field.

'M128': 1 byte value. 0 in modulo 8 operating mode (1 byte control field), 1 in modulo 128 operating mode (2 byte control field). For correct decoding of 'ctrl' above.

Decode_U_Frame_BASIC ()

Decode_U_Frame_BASIC is called by the LLC interrupt server to transfer a received HDLC U frame to a higher layer protocol software.

Following information is passed to Decode_U_Frame_BASIC:

'pei': (refer to Decode_S_Frame).

'sapi': (refer to Decode_S_Frame).

'tei': (refer to Decode_S_Frame).

'ctrl': 1 byte value representing the contents of the received HDLC control field.

PassLongFrame_BASIC ()

PassLongFrame_BASIC is called by the LLC interrupt server to transfer received HDLC I and UI frames to a higher layer protocol software.

The LLC passes a pointer to a structure (FRAME_PASS) containing information about the received frame to PassLongFrame_BASIC. Please refer to the following paragraph for a description of this structure.

6.3 LLC Code Elements

6.3.1 Structures

The Structure 'ISAC'

As the various routines in the LLC require facilities to store information about the device they control, the global variable 'pt' of the type 'ISAC' has been introduced. The type 'ISAC' is a structure containing imperative information elements. These information elements are listed below:

Status Information

pt_op_mode	operating mode of the ISAC-S HDLC controller (auto-mode, non-auto-mode...)
pt_state	Flags of 'pt_state' indicate the various device states.

I/O buffer related elements

These elements are used when the HDLC data is transmitted or received. In both the transmit and receive directions additional RAM is required to store data on an intermediate basis. This buffer will be referred to as the data frame. Related information is stored in the following elements:

Transmit buffer pointers

pt_tx_start	pointer to the starting point of the data frame for transmission
pt_tx_curr	pointer to the present byte to be sent

Receive buffer pointers

pt_rx_start	pointer to the starting point of the receive data frame.
pt_rx_curr	pointer to the next free position in the receive buffer.

Data byte counters

pt_tx_cnt	number of bytes yet to be transmitted
pt_rx_cnt	number of bytes currently received

The following elements are used to store the type of frame:

pt_rx_frame	type of received frame.
pt_tx_frame	type of transmitted frame.

The Structure 'FRAME_PASS'

The variable 'fp' of the type FRAME_PASS is used when the LLC interrupt server has received a valid HDLC I or UI frame. A pointer to 'fp' is passed to PassLongFrame_BASIC. FRAME_PASS contains all information about the received HDLC frame. Following elements are used:

mmu_buff	start of MMU buffer which is used for the temporary storage of that HDLC frame.
start_of_i_data	Start of the I data field in this MMU buffer.
i_data_cnt	Number of bytes in the I data field.
Two_byte_cf	0 for a one byte HDLC control field, 1 for a two byte HDLC control field.
ctrl_field	HDLC control field.
pei	1 byte value identifying the performed address recognition. The bits 0, 1 and 2 of 'pei' represent the bits TA, SA0 and SA1 of the ISAC-S' RSTA register.
frame	Type of HDLC frame; 0 = I-frame, 1 = UI-frame.
sapi	Received HDLC SAPI address byte. Bit 1 of 'sapi' is the C/R bit value (RSTA:CR). The most significant 6 bits of 'sapi' are 0 in auto-mode, non-auto-mode and transparent mode.
tei	Received HDLC TEI address byte. 'tei' is 0 in auto-mode and non-auto mode.

6.3.2 Definitions and Naming Conventions

Public functions are declared with an EXPORT (only for better readability). External functions are imported using an IMPORT which is the redefinition of C's 'extern'. Any function which is only used locally is declared with a LOCAL (= 'static').

6.3.3 Type Definitions

For reference here is a list of the type definitions used in the LLC's.

type definitions	meaning
BYTE	one byte value
WORD	word = two byte value
FPTR	far pointer to BYTE

6.3.4 Macro Definitions

Error conditions and other states of the ISAC-S must be reported to higher layers. This reporting is realized by a few macros which are executed when such conditions are detected. These macros can be mapped to any form of message a higher layer software requires. Any kind of immediately necessary actions may be defined in those macros as well. By using such constructs the code can be kept compact and clearly readable.

Layer 1 Related Status Message

DECODE_L1_STATUS	for L1 status (IC channel indication) decoding.

HDLC Controller Related Status and Error Messages

CRC_ERROR	CRC error.
MISSING_ACKNOWLEDGE	A 'Missing HDLC I-frame acknowledge' is generated when an acknowledge message for a previously sent I-frame is outstanding and the HDLC message transfer mode is changed from auto-mode to non-auto-mode. An outstanding acknowledge is indicated by the ISAC-S in register STAR2 ('timer recovery status' and 'waiting for acknowledge' bits).
MMU_ERROR	No memory available to store incoming frame.
N201_ERROR	N201 error, HDLC frame is too long.
PEER_REC_READY	Peer receiver ready.
PEER_REC_BUSY	Peer receive busy.
PROTOCOL_ERROR	Protocol error (PCE interrupt).
REC_FRAME_OVERFLOW	Receive frame overflow.
REC_DATA_OVERFLOW	Receive data overflow (RDO interrupt).
REC_ABORTED	Receive aborted (RAB interrupt).
TX_ACKNOWLEDGE	Transmit frame acknowledge.
TIN_ERROR	TIN interrupt, status enquiry.
TX_DATA_UNDERRUN	Transmit data underrun (XDU interrupt).
XMR_ERROR	Transmit message repeat indication (XMR interrupt).

Following macros are used when a 'timer recovery status' (register STAR2, bit TREC) is recognized.

ENABLE_TREC_STATUS_CHECK	enable 'timer recovery status' check procedure.
DISABLE_TREC_STATUS_CHECK	disable 'timer recovery status' check procedure.

6.4 Interrupts

Int_ICC is to be called in the case of ISAC-S interrupts. The following interrupts are handled directly in Int_ICC:

'Transmit pool ready' interrupt (ISTA:XPR)

'Timer' interrupt (ISTA:TIN).

'Receive Status Change' interrupt (ISTA:RSC).

'Extended' interrupt (ISTA:EXI).

The 'Receive Pool Full' (ISTA:RPF) and 'Receive Message End' (ISTA:RME) interrupts are handled by function RX_ICC. The 'CI or SQ channel change' interrupt (ISTA:CISQ) is handled by IntL1_SBC.

Please note that the following interrupts are not handled by the interrupt service routine described here:

ISTA:SIN (synchronous transfer interrupt)
EXIR:SOV (synchronous transfer overflow)
EXIR:MOS (MONITOR status)
EXIR:SAW (subscriber awake)
EXIR:WOV (watchdog timer overflow)

6.5 LLC Routine Reference

6.5.1 ISAC®-S Layer-1 Functions: The SBC Part

ActL1_SBC ()

Initiates layer-1 activation. The appropriate CI code (activate request) is written to the CI channel if the layer 1 is not already activated. ActL1_SBC then returns with ACK_DONE. The subsequent status changes of the SBC will cause CI channel status change (CISQ) interrupts and these will be evaluated in the layer-1 interrupt service routine IntL1_SBC.

If the layer 1 is already activated nothing is carried out but ActL1_SBC calls DECODE_L1_STATUS to report the activated state.

ArL1_SBC ()

Activates a local loop in the SBC. The appropriate CI code (activate request loop) is written to the SBC. ArL1_SBC returns with ACK_DONE. The subsequent status changes of the SBC will generate CISQ interrupts and these will be evaluated and reported in the layer-1 interrupt service routine IntL1_SBC.

EnaClk_SBC ()

EnaClk_SBC enables clocking in TE configurations when the SBC is in power down state. If first tests if clocks are actually there. If there are clocks the function returns with FALSE. If there are no clocks (power down state) the power-up procedure is implemented. The SPU bit in register SPCR is set. The TIM code is written to the CI channel. EnaClk_SBC waits until the power up state (PU) is indicated before the SPU bit is reset to 0. The routine then returns with TRUE.

InitL1_SBC ()

Initializes the layer-1 controller and resets the SBC (ResL1_SBC). Timing mode 0 is set and the TIC bus address is also programmed. Finally ResL1_SBC is called to reset the SBC.

ResL1_SBC ()

This routine resets the SBC. It also checks that the SBC is functioning correctly.

Reset procedure:

A software reset command (RS) is sent to the SBC via the IOM CI0 channel. ResL1_SBC waits for the expected new state (EI) if no timeout condition occurs and issues a release command (DIU).

If the new state (EI) is not observed the SBC will be deemed to be defective.

IntL1_SBC ( ) **Interrupt Handler**

Handles the CISQ interrupts which indicate changes in the SBC status. The final confirmation of deactivation is carried out here. The actual SBC state is evaluated by reading register CIR0. The following is then carried out:

If the CI channel indication is 'pending deactivation' state (DR), DIU is sent to deactivate the SBC.

If the indication is an 'activation indication' (AI) the activation must be confirmed from the TE side. IntL1_SBC does it automatically by writing an 'activation request' (AR). In this way this requirement of the ISAC-S is transparent to the higher protocol layers.

After every CI channel status change interrupt (CISQ) DECODE_L1_STATUS is called to report the current layer-1 state.

6.5.2 ISAC®-S HDLC Controller Related Functions: The ICC Part

InitPeitab_ICC ()

Initializes the local variable 'pt'. InitPeitab_ICC is to be called once during the system initialization phase.

InitLay2_ICC ()

Initializes the HDLC controller. The function arguments allow the selection of the HDLC controller message transfer mode (auto-mode, non-auto-mode, ...), one or two byte HDCL control field operation (modulo 8 or 128) and the setting of the ISAC-S internal hardware timer.

After InitLay2_ICC is called the TEI values for a Broadcast Link are programmed (TEI = FF hex). The HDLC controller is not reset.

StoreTEI_BASIC ()

StoreTEI_BASIC is used to program a TEI value in register TEI1 or TEI2 depending on the function argument value.

StoreSAPI_BASIC ()

StoreSAPI_BASIC is used to program a SAPI value in register SAP1 or SAP2 depending on the function argument value.

RecReady_BASIC ()

Sets HDLC receiver ready or not ready depending on the function argument value.

ResetHDLC_BASIC ()

ResetHDLC_BASIC resets the HDLC controller. Status flags of the local variable 'pt' indicating any on-going data transmissions or receptions are reset and memory buffers are released.

SendFrame_ICC ()

SendFrame_ICC initiates the transmission of HDLC frames (S, U, I, UI frames).

A frame can not be sent if the transmit path is still in use, i.e. if the previous transmission is not finished, if the timer recovery state is indicated (only for I frames) or if the XFIFO is blocked (STAR:XFW bit).

If the transmission is begun the interrupt handler (Int_ICC) will handle subsequent tasks, for example shifting remaining data bytes into the XFIFO or calling the MMU to release the memory buffer.

Loop_ICC ()

Switches testloop at the IOM interface on or off, i.e. connects internally the data upstream and data downstream lines. This is achieved through setting/resetting the TLP bit in register SPCR. If the layer-1 part does not deliver clocks while in the deactivated state the clocks will be enabled when the loop is switched on by means of EnaClk_SBC. When the loop is switched off the layer-1 part will return to its normal deactivated state.

Int_ICC () Interrupt Handler

Evaluates and handles the ISAC-S interrupts.

Interrupt service procedure:

The bits of the interrupt status register ISTA are scanned. XPR, TIN, RSC, and EXI interrupts are handled directly by Int_ICC. For RPF and RME interrupts the function RX_ICC is called, for CISQ interrupts IntL1_SBC is called. The interrupt related actions performed are:

- XPR(transmit pool ready) interrupt, but no TIN and no PCE (EXIR:PCE) interrupt:
  a) HDLC controller reset was given previously.
  b) last transmission is finished. The XFIFO will be loaded if there are more bytes to be sent. If not, a 'transmit frame acknowledge' can be generated (if depends on the message transfer mode and some other conditions).
- TIN interrupt:
  The HDLC controller's internal timer has expired (in auto-mode only).
- RSC (receiver status change of remote station) interrupt:
  A status change of the remote station's receiver has been detected. This is reported to the higher layers.
- EXI (extended) interrupt:
  One of the six non-critical interrupts has been generated. The exact cause is read from register EXIR and reported to the higher layers.

RX_ICC () Interrupt Handler

Handles the receive pool full and receive message end (RPF and RME) interrupts if TIN and PCE (EXIR:PCE) interrupt are not indicated. Received frames are handed over to the higher software levels. Errors detected during the frame reception are reported to the higher layers.

RPF interrupt: 32 data bytes are in the RFIFO. The end of the received frame is yet to be received and the message is not complete.

RME interrupt: The receive message is complete. The RFIFO contains the last bytes of a frame greater than 32 bytes long or a complete frame. In the case of a long frame the beginning of this frame will already have been received using the RPF interrupt. Address and control field information is examined, the type of frame (HDLC U, UI, I or S-frame) is determined and the validity of the frame is checked. Finally the frame or a error condition message is sent to the higher layers.

Check_TREC_status_ICC ()

Check_TREC_status_ICC () is called periodically by the operating system, if 'timer recovery status' (STAR2:TREC) was detected during a previous XPR interrupt handling. A 'transmit frame acknowledge' for an HDLC I-frame is generated if the TREC status is left and no timer interrupt (ISTA:TIN) is indicated.

6.6 Listing of Driver Routines

```
/*****************************************************************************/
/*                                                                           */
/*    Copyright (C) Siemens AG, 1991. All rights reserved.                   */
/*    ====================================================                   */
/*                                                                           */
/*    File       :  icc.c                                                    */
/*    Function   :  ICC / ISAC-S driver routines                             */
/*    Created    :  Jan. 1991                                                */
/*    Compiler   :  Microsoft(R) C Optimizing Compiler, version 5.1          */
/*                                                                           */
/*****************************************************************************/



      /*  Include Files                                                      */
      /*  =============                                                      */

#include "def.h"                  /*  type redefinitions                   */

#include "icc.h"                  /*  ICC / ISAC-S specific                */
                                  /*  definitions                          */

#include "message.h"              /*  macros for status and error          */
                                  /*  messages                             */



      /*  Import Functions                                                   */
      /*  ================                                                   */

      /*  from crt0.asm                                                      */
IMPORT    void                    STRING_IN();
IMPORT    void                    STRING_OUT();

      /*  from sbc.c                                                         */
IMPORT    void                    IntL1_SBC();
IMPORT    int                     EnaClk_SBC();

      /*  from basic_I2.c                                                    */
IMPORT    void                    PassLongFrame_BASIC ();
IMPORT    void                    Decode_S_Frame_BASIC ();
IMPORT    void                    Decode_U_Frame_BASIC ();

      /*  from mmu.c                                                         */
IMPORT    int                     MMU_free ();
IMPORT    FPTR                    MMU_req ();
```

```
      /*  Export Functions                                              */
      /*  ================                                              */

EXPORT    void                Check_TREC_status_ICC (void);
EXPORT    int                 InitLay2_ICC (BYTE,BYTE,BYTE);
EXPORT    void                InitPeitab_ICC (void);
EXPORT    void                Int_ICC (void);
EXPORT    int                 Loop_ICC (BOOLEAN);

EXPORT    int                 RecReady_ICC (BYTE);
EXPORT    int                 ResetHDLC_ICC (void);
EXPORT    int                 StoreTEI_ICC (BYTE,BYTE);
EXPORT    int                 StoreSAPI_ICC (BYTE,BYTE);
EXPORT    int                 SendFrame_ICC (BYTE,WORD,FPTR);


      /*  Local Functions                                               */
      /*  ===============                                               */

LOCAL     void                RX_ICC (BOOLEAN);


      /*  Variables                                                     */
      /*  ===============                                               */

GLOBAL    ISAC   pt[1];


      /*  Function Declarations                                         */
      /*  =====================                                         */
/*********************************************************************/
/*                                                                   */
/*        Function  :InitPeitab_ICC ()                               */
/*        Parms     :none                                            */
/*        purpose   :initialization of locally used variables        */
/*                    and structure elements                         */
/*                                                                   */
/*********************************************************************/

EXPORT void
InitPeitab_ICC ()
{
      pt→pt_state = STATE_TX_MMU_FREE;
      DISABLE_TREC_STATUS_CHECK ();
}
```

```
/*****************************************************************/
/*                                                               */
/*        Function      :InitLay2_ICC ()                         */
/*        Parameters :                                           */
/*                                                               */
/*            'modulo'       0  modulo 8 operation               */
/*                           1  modulo 128 operation             */
/*                                                               */
/*            'mode'         operating mode.(auto-mode,non-auto-mode,*/
/*                           etc.)                               */
/*                                                               */
/*            'tim_mode' value for the TIMR register (valid in auto */
/*                           mode only) refer to the description of  */
/*                           that register in the data sheets.   */
/*                                                               */
/*        Purpose:        HDLC controller initialization         */
/*                           After execution of InitLay2_ICC, the    */
/*                           TEI values for the Broadcast  Link are  */
/*                           programmed.                         */
/*                                                               */
/*        Note:      No HDLC controller reset is done.           */
/*                   Only two bytes address fields are supported */
/*                                                               */
/*            If the ICC (ISAC) is reprogrammed from auto-mode to   */
/*            NON-auto-mode the successful transmission and         */
/*            acknowledgement of an I-frame currently sent is not   */
/*            assured.                                           */
/*            Switching from auto-mode to non-auto-mode causes an   */
/*            I-frame to be transmitted completely by the ICC. But  */
/*            the XPR interrupt in non-auto-mode only indicates     */
/*            that the frame was sent out of the XFIFO. It          */
/*            indicates not the successful transmission of the      */
/*            I-frame as it is in auto-mode (timer super-vision,    */
/*            polling for acknowledge frames)!                   */
/*            Therefore if an I-frame acknowledge is outstanding and*/
/*            the mode is changed from auto-mode to NON-auto-mode   */
/*            MISSING_ACKNOWLEDGE is called to generate a warning   */
/*            message.                                           */
/*            MISSING_ACKNOWLEDGE is also called if 'timer          */
/*            recovery' status (TREC) or 'waiting for acknowledge   */
/*            (WFA)' is indicated.                               */
/*                                                               */
/*****************************************************************/

EXPORT int
InitLay2_ICC (modulo, mode, tim_mode)
       BYTE      modulo, mode, tim_mode;
{
```

```
BYTE    mode_reg;

if (modulo != 0 && modulo != 1)
       return (ACK_WRONG_PARM);

outp (ISAC_R_MASK, 0xFF);    /* mask out all device interrupt    */

mode_reg =inp (ISAC_R_MODE) & (MODE_HMD2|MODE_HMD1|MODE_HMD0);

switch (mode)                /* select OPERATING MODE            */
{                            /* =====================            */

      case MODE_AUTO:        /* HDLC auto-mode                   */
                             /*   full address recognition,      */
                             /*   internal timer mode, receiver  */
                             /*   active, 2 bytes address fields */
                             /*   are selected.                  */
            mode_reg    |=(MODE_TMD|MODE_RAC|MODE_ADM);
            outp (ISAC_R_TIMR, tim_mode);
            break;

      case MODE_NON_AUTO:    /* HDLC non-auto-mode               */
                             /*   full address recognition,      */
                             /*   receiver active, 2 byte address*/
                             /*   fields                         */
            mode_reg      |=(MODE_MDS0 | MODE_RAC | MODE_ADM);  */

            if (inp(ISAC_R_STAR2) & (STAR2_TREC | STAR2_WFA))
            {
                    MISSING_ACKNOWLEDGE (PEI);
                    ResetHDLC_ICC ();
            }

            outp (ISAC_R_TIMR, 0);
            break;

      case MODE_TRANSP:      /* TRANSPARENT MODE                 */
                             /* SAPI-address (high-byte)         */
                             /* recognition                      */
            mode_reg    |=(MODE_MDS1|MODE_MDS0|MODE_RAC|
                        MODE_ADM);
            break;

      case MODE_EXT_TRANSP: /* EXTENDED TRANSPARENT MODE         */
                             /* no address recognition           */
            mode_reg     |=(MODE_MDS1|MODE_MDS0|MODE_RAC);
            break;
```

```
        default:
                outp (ISAC_R_MASK, 0x00);
                return (ACK_WRONG_PARM);
    }

    pt→pt_op_mode = mode;          /* save MODE register settings      */

                                   /* modulo: 1 (mod 128); 0 (mod 8)   */
    outp (ISAC_R_SAP2, (BYTE) (modulo ? 0x02 : 0x00));
    outp (ISAC_R_TEI2, 0xFF);

    if (modulo)
          pt→pt_state|= STATE_M128;
    else
          pt→pt_state & = ~STATE_M128;

                                   /* output MODE register settings    */
                                   /* and demask interrupts            */
    outp (ISAC_R_MODE, mode_reg);
    outp (ISAC_R_MASK, 0x00);

    return (ACK_DONE);
}
/*****************************************************************************/
/*                                                                           */
/*    Function    :StoreTEI_ICC ()                                           */
/*    Parms       :'tei' and 'reg2'                                          */
/*    purpose     :program TEI in register TEI1 (reg2=0) or                  */
/*                 TEI2 (reg2=1)                                             */
/*                                                                           */
/*****************************************************************************/

EXPORT int
StoreTEI_ICC (tei, reg2)
    BYTE tei, reg2;
{
    if (reg2 == 1)                 /* store TEI in register TEI2       */
          outp (ISAC_R_TEI2, tei);
    else
    {                              /* store TEI in register TEI1       */
          outp (ISAC_R_XAD2, tei);
          outp (ISAC_R_TEI1, tei);
    }
    return (ACK_DONE);
}
```

```
/*****************************************************************/
/*                                                               */
/*    Function   :StoreSAPI_ICC ()                               */
/*    Parms      :sapi, reg2                                     */
/*    purpose    :store SAPI in register SAPI1 (reg2 = 0) or     */
/*                SAPI2 (reg2 = 1)                               */
/*                                                               */
/*****************************************************************/

EXPORT int
StoreSAPI_ICC (sapi, reg2)
    BYTE                    sapi, reg2;
{
    sapi & ~0x03;

    if (reg2 == 1)              /* store SAPI in SAP2            */
            outp (ISAC_R_SAP2, sapi|((pt->pt_state & STATE_M128)
            ? 0x02 : 0x00));
    else
    {                           /* store SAPI in SAP1            */
            outp (ISAC_R_XAD1, sapi);
            outp (ISAC_R_SAP1, sapi);
    }
    return (ACK_DONE);
}
/*****************************************************************/
/*                                                               */
/*    Function   :RecReady_ICC ()                                */
/*    Parms      :busy                                           */
/*    purpose    :set HDLC receiver ready      ('ready' = 1)     */
/*                              not ready       ('ready' = 0)     */
/*                              To be used in auto-mode only      */
/*                                                               */
/*****************************************************************/

EXPORT int
RecReady_ICC (ready)
    BYTE        ready;
{
    outp (ISAC_R_CMDR, (BYTE) (ready ? 0x00 : CMDR_RNR));
    return (ACK_DONE);
}
/*****************************************************************/
/*                                                               */
/*    Function   :ResetHDLC_ICC ()                               */
/*    Parms      :none                                           */
/*    purpose    :reset HDLC controller                          */
/*                                                               */
```

```
/*****************************************************************************/

EXPORT int
ResetHDLC_ICC ()
{
    outp (ISAC_R_MASK, 0xFF);

                                /*  clear receive and transmit      */
                                /*  paths, i.e. clear the status    */
                                /*  variables indicating any        */
                                /*  transmission or reception of    */
                                /*  frames and release the MMU      */
                                /*  buffers                         */
    FREE_TX_PATH (PEI);

    if (pt->pt_rx_start)
    {
        MMU_free (pt->pt_rx_start);
        pt->pt_rx_start          =NULL_PTR;
        pt->pt_state             &= ~STATE_REC_ACTIVE;
        pt->pt_rx_frame          = 0x00;
        pt->pt_rx_cnt            = 0;
    }

    pt-pt_state      &= ~STATE_REC_ACTIVE;

                               /*  set the reset flag in the state  */
                               /*  variable. This allows the        */
                               /*  interrupt service routine to     */
                               /*  react correctly on the following */
                               /*  XPR interrupt                    */
    pt->pt_state|= STATE_HDLC_RESET;
                               /*the reset commands:                 */
                               /*- receive message complete (RME)    */
                               /*- reset hdlc receiver      (RHR)    */
                               /*- transmitter reset        (XRES)   */
    outp (ISAC_R_CMDR, CMDR_RMC|CMDR_RHR|CMDR_XRES);

    if (pt->pt_op_mode         /*write TIMR register to stop the     */
    == MODE_AUTO)
                               /*internal timer in auto-mode
*/
        outp (ISAC_R_TIMR, inp(ISAC_R_TIMR));

    outp (ISAC_R_MASK, 0);
    return (ACK_DONE);
}
```

```
/*****************************************************************/
/*                                                               */
/*        Function   :   SendFrame_ICC ()                        */
/*        Parms      :   'frame_type'    specifying the frame    */
/*                       'cnt'           number of bytes to send */
/*                       'frame_ptr'     pointer to the data bytes*/
/*        purpose    :   Initiate transmission of HDLC frames    */
/*                       (S, U, I, UI)                           */
/*                                                               */
/*****************************************************************/

EXPORT int
SendFrame_ICC (frame_type, cnt, frame_ptr)
    BYTE            frame_type;
    WORD            cnt;
    FPTR            frame_ptr;

    BYTE            cmd;

                            /* return if XFIFO is not write     */
                            /*   enable                         */
    if (!(inp (ISAC_R_STAR) & 0x40))
            return (ACK_ACCESS_FAULT);

                            /*   return if transmit path still  */
                            /*   blocked and not in auto-mode   */
    if (pt->pt_state & STATE_TX_ACTIVE && pt->pt_op_mode != MODE_AUTO)
            return (ACK_ACCESS_FAULT);

    if (pt->pt_op_mode == MODE_AUTO)
    {
                            /*   it is not allowed to send an I */
                            /*   frame in the timer recovery    */
                            /*   or in waiting_for_acknowledge  */
                            /*   status                         */
            if (inp(ISAC_R_STAR2) & (STAR2_TREC|STAR2_WFA))
                    if (frame_type == FRAME_I)
                            return (ACK_ACCESS_FAULT);

            if (inp(ISAC_R_STAR2) & STAR2_WFA)
                    if (pt->pt_state & STATE_TX_MMU_FREE)
                            {
                                MMU_free (pt->pt_tx_start);
                                pt->pt_state & = ~STATE_TX_MMU_FREE;
                            }
    }
```

```
pt→pt_state|= STATE_TX_    /*transmitter is active                */
ACTIVE;
pt→pt_tx_start = frame_ptr; /*store data frame pointer             */
pt→pt_tx_frame = frame_type;/*and frame type                       */

if (cnt <=32)
{
                           /*if the number of bytes is <=32       */
                           /*the frame can be shifted             */
                           /*completely into the XFIFO            */
    STRING_OUT (frame_ptr, ISAC_R_FIFO, cnt);
    pt→pt_tx_cnt = 0;
}
else
{
                           /*if the number of bytes is            */
                           /*32 the first 32 are shifted          */
                           /*into the XFIFO; the interrupt        */
                           /*service routine handles the          */
                           /*rest                                 */
     STRING_OUT (frame_ptr, ISAC_R_FIFO, 32);
     pt→pt_tx_cnt = cnt - 32;
     pt→pt_tx_curr = frame_ptr + 32;
}

                           /*compute the transmission command     */
                           /*to be entered into the CMDR          */
                           /*register                             */
                           /*The 'transmit I frame' command       */
                           /*(XIF) must be used in auto-mode      */
                           /*when it is an HDLC I frame.          */
                           /*The 'transmit transparent frame'     */
                           /*command (XTF) must be used in        */
                           /*all other cases                      */
if (pt→pt_op_mode == MODE_AUTO)
{
     cmd = (pt→pt_tx_frame == FRAME_I) ? CMDR_XIF : CMDR_XTF;

     if (inp (ISAC_R_STAR) & CMDR_RNR)
          cmd|= CMDR_RNR;
}
else
     cmd = CMDR_XTF;
                           /*When the frame fits completely       */
                           /*into the XFIFO (<= 32 data bytes)    */
                           /*the XME command must be given        */
```

```
    if (!pt→pt_tx_cnt)
            cmd|= CMDR_XME;
    outp (ISAC_R_CMDR, cmd);      /*output command byte to CMDR        */


                                  /*  A flag is set if an UI frame (ID*/
                                  /*  check response) is sent while    */
                                  /*  the HDLC controller is waiting   */
                                  /*  for an I frame acknowledge (in   */
                                  /*  auto-mode only)                  */
                                  /*  The interrupt service routine    */
                                  /*  (Int_ICC) checks that flag       */
                                  /*  when handling a XPR interrupt    */
    if (inp(ISAC_R_STAR2) & STAR2_WFA && pt→pt_op_mode == MODE_AUTO
                                  && frame_type == FRAME_UI)
            pt→pt_state|= UI_SENT_WHILE_WAITING_FOR_ACK;

    return (ACK_DONE);
}
/*********************************************************************/
/*                                                                   */
/*          Function  :Loop_ICC ()                                   */
/*          Parms     :    'on'      1   -test-loop on               */
/*                                     0   -test-loop off              */
/*          purpose   :activates / deactivates a testloop at the     */
/*                     IOM interface                                 */
/*                                                                   */
/*********************************************************************/

EXPORT int
Loop_ICC (on)
    BOOLEAN         on;
{
    BYTE      r_spcr;
    if (on)                       /*Loop ON                          */
{
            pt→pt_state|= STATE_LOOP;

                                  /*  dummy value in the             */
                                      cixr register
                                  /*  prevents a false               */
                                      interpretation of
                                  /*  the incoming (looped)          */
                                      C/I channel
            if (EnaClk_SBC ())
                    outp (ISAC_R_CIX0, 0×6F);

            r_spcr = inp (ISAC_R_SPCR);
```

```
        outp (ISAC_R_SPCR, r_spcr | SPCR_TPL);
    }
    else                              /* Loop OFF                    */
    {
        r_spcr = inp (ISAC_R_SPCR) & ~SPCR_TPL;
        outp (ISAC_R_SPCR, r_spcr);

        pt→pt_state & = ~STATE_LOOP;
    }

    return (ACK_DONE);
}
  /* ** The interrupt service routines  ***                          */
  /*****************************************************************/


/*****************************************************************/
/*                                                                */
/*      Function    :Int_ICC ()                                   */
/*      Parms       :none                                         */
/*      purpose     :handle ICC (ISAC-S, ISAC-P) interrupts       */
/*                                                                */
/*****************************************************************/

EXPORT void
Int_ICC ()
{
    WORD                          cnt;
    BYTE                     exir, cmd;
    register   BYTE ista;

    if ((ista = inp (ISAC_R_ISTA)) == 0x00)
            return;

    exir = inp (ISAC_R_EXIR);


                                  /*     XPR interrupt             */
                                  /*     =============             */
                                  /* the XPR interrupt indicates   */
                                  /* the XFIFO ready status.       */
                                  /* Reasons:                      */
                                  /* - HDLC controller reset       */
                                  /* - data transmission finished  */
```

```
 if ((ista & ISTA_XPR) && !(ista & ISTA_TIN) && !(exir &EXIR_PCE))
{
                              /* transmit byte counter is 0        */
                              /*    ----------------------         */
    if ((cnt = pt→pt_tx_cnt) == 0)
    {
                              /* HDLC controller reset command     */
                              /* given previously ?                */
                              /*    -------------------------      */
                              /* do nothing when it was a HDLC     */
                              /* controller reset. Only the        */
                              /*   indicating flag must be cleared*/
        if (pt→pt_state & STATE_HDLC_RESET)
              pt→pt_state & = ~STATE_HDLC_RESET;
        else
        {
                              /* XPR was generated because the     */
                              /* last transmission is finished     */
                              /*    -------------------------  */
                              /*    auto-mode operation ?*/
             if (pt→pt_op_mode == MODE_AUTO)
             {
                              /*    UI frame sent while waiting    */
                                    for
                              /*    I frame acknowledge ?          */
                if (pt→pt_state & UI_SENT_WHILE_WAITING_FOR_ACK)
                {
                              /*    the UI frame was sent out      */
                                    if the
                              /*    XFIFO is empty  (write enable)*/
                  if (inp(ISAC_R_STAR) & STAR_XFW)
                          TX_ACKNOWLEDGE (PEI, pt→pt_tx_frame);

                  pt→pt_state & = ~UI_SENT_WHILE_WAITING_FOR_ACK;

                              /*    if we are in timer recovery    */
                              /*    status the TREC status check   */
                              /*    procedure is activated. The    */
                              /*    transmit acknowledge for the I*/
                              /*    frame must not be generated!!!*/
                  if (inp (ISAC_R_STAR2) & STAR2_TREC)
                              ENABLE_TREC_STATUS_CHECK ();
                  else
                              TX_ACKNOWLEDGE (PEI, (BYTE) FRAME_I);
             }
             else
             {
```

```
                                /*  if we are in timer recovery      */
                                /*  status and the last frame was an*/
                                /*  I frame the TREC status check    */
                                /*  procedure is activated.          */
                                /*  If not an transmit acknowledge   */
                                /*  is generated                     */
                        if (pt→pt_tx_frame == FRAME_I &&
                                (inp (ISAC_R_STAR2) & STAR2_TREC))
                                ENABLE_TREC_STATUS_CHECK ();
                        else
                                TX_ACKNOWLEDGE (PEI, pt→pt_tx_frame);
                }
        }
        else
                                /*  In all other operating modes    */
                                /*  (non-auto-mode, transparent
                                    mode,
                                /*  ...) the transmit acknowledge   */
                                /*  can be generated at once.       */
           TX_ACKNOWLEDGE (PEI, pt→pt_tx_frame);

                                /*  transmit byte counter and       */
                                    status
                                /*  flag are reset.                 */
                                /*  The MMU buffer used for         */
                                /*  temporary data storage is       */
                                /*  released if necessary           */

        pt→pt_tx_cnt = 0;
        pt→pt_state & = ~STATE_TX_ACTIVE;

        if (pt→pt_state & STATE_TX_MMU_FREE)
        {
            MMU_free (pt→pt_tx_start);
            pt→pt_state & = ~STATE_TX_MMU_FREE;
        }
    }
}
else
{                               /* transmit counter is not 0,       */
                                /*      more data is to be sent!    */
                                /*      ------------------          */
if (pt→pt_op_mode == MODE_AUTO)
        cmd = (pt→pt_tx_frame ? CMDR_XTF : CMDR_XIF) |
                (inp(ISAC_R_STAR) & CMDR_RNR);
else
        cmd = CMDR_XTF;
```

```
            if (cnt <= 32
            {
                                /*  if there are than 32 bytes left */
                                /*  they all can be shifted         */
                                /*  completely into the XFIFO. The  */
                                /*  XME command can be issued       */
                STRING_OUT (pt→pt_tx_curr, ISAC_R_FIFO, cnt);
                outp (ISAC_R_CMDR, cmd | CMDR_XME);
                pt→pt_tx_cnt     = 0;
            }
            else
            {                   /*  more than 32 bytes are left to  */
                                /*  be sent; write 32 into the XFIFO*/
                STRING_OUT (pt-pt_tx_curr, ISAC_R_FIFO, 32);
                outp (ISAC_R_
                CMDR, cmd);     /*  give the transmit command,      */
                pt→pt_tx_curr
                + = 32;         /*  update current buffer pointer   */
                pt→pt_tx_cnt
                - = 32;         /*  and counter for remaining bytes */
            }
        }
    }

    if (ista & ISTA_TIN)        /*  TIN interrupt                   */
    {                           /*  =============                   */
                                /*  reset the HDLC controller and   */
                                /*  status variables and generate an*/
                                /*  error message                   */
        pt→pt_state & = ~UI_SENT_WHILE_WAITING_FOR_ACK;
        ResetHDLC_ICC ();
        DISABLE_TREC_STATUS_CHECK ();
        TIN_ERROR (PEI);
    }
                                /*  HDLC receiver interrupt?        */
                                /*  ========================        */
                                /*  (receive pool full or receive   */
                                /*  message end and not PCE and not */
                                /*  TIN)                            */
    if ((ista & (ISTA_RPF | ISTA_RME))
                          && !(exir & EXIR_PCE) && !(ista & ISTA_TIN))
            RX_ICC (ista & ISTA_RPF);


                                /*  status change of the remote     */
                                /*  station's receiver              */
                                /*  (i.e. RR or RNR received).      */
                                /*  The status can be determined by */
```

```
                                /* reading the RRNR bit of          */
                                /* register STAR                    */
    if (ista & ISTA_RSC)
    {
         if (inp (ISAC_R_STAR) & 0x10)
           PEER_REC_BUSY (PEI); /*   peer receiver busy             */
         else
           PEER_REC_READY (PEI);/*   peer receiver ready            */
    }


                                /* B (2.x) versions of L1 device    */
                                /* controllers can't prevent CIC bit*/
                                /* being set even when masked.      */
                                /* CIC interrupt ? (layer-1 device  */
                                /*                  status change)  */
    if (ista & ISTA_CIC)
           IntL1_SBC ();

    if (ista & ISTA_EXI)        /*  Extended interrupt?             */
    {                           /*  ===================             */

                                /*  transmit message repeat int.?   */
           if ((exir & EXMIR_XMR) && !(exir & EXIR_PCE) && !
           (ista & ISTA_TIN))
           {
                   XMR_ERROR (PEI);
                   FREE_TX_PATH (PEI);
           }

           if (exir & EXIR_XDU) /*  transmit data underrun?         */
           {
                   TX_DATA_UNDERRUN (PEI);
                   FREE_TX_PATH (PEI);
           }

           if (exir & EXIR_PCE) /*  protocol error interrupt?       */
           {
                   PROTOCOL_ERROR (PEI);
                   ResetHDLC_ICC ();
           }


           if (exir & EXIR_RFO) /*  receive frame overflow int.?    */
           {
                   REC_FRAME_OVERFLOW (PEI);
                   MMU_free (pt→pt_rx_start);
```

```
            pt→pt_rx_start    = NULL_PTR;
            pt→pt_state&      = ~STATE_REC_ACTIVE;
            pt→pt_rx_frame    = 0;
            pt→pt_rx_cnt      = 0;
        }

        if (exir & EXIR_MOR)     /*  MON channel interrupt?          */
        {
                outp (ISAC_R_MOCR, 0xAA);
                while (inp
                (ISAC_R_MOCR))  /*  clear out MOSR                  */
                     ;
                outp (ISAC_R_MOCR, 0x00);
        }
    }
}
/*****************************************************************/
/*                                                               */
/*      Function    :RX_ICC ()                                   */
/*      Parms       :'rpf' = 1 if RPF interrupt                  */
/*      purpose     :handle HDLC receiver interrupts             */
/*                               ICC (ISAC-S, ISAC-P)            */
/*                                                               */
/*****************************************************************/

LOCAL void
RX_ICC (rpf)
    BOOLEAN                      rpf;
{
    WORD                         RecCnt, ctrl;
    FPTR                 ptr;
    BYTE                 pei = PEI;
    BYTE                 rsta, tei, sapi, frame_status = VALID;
    BOOLEAN              Two, AutoM;

                                 /* RPF interrupt:                 */
                                 /* 32 bytes of a frame longer than */
                                 /* 32 bytes have been received    */
                                 /* and are now available in the   */
                                 /* RFIFO.                         */
                                 /* The message is not complete.   */
    if (rpf)
            RecCnt = 32;
    else
    {
                                 /* RME interrupt:                 */
```

```
                                /* Receive message end. The RFIFO    */
                                /* contains a complete frame         */
                                /* (length <= 32 byte) or the last   */
                                /*    bytes or a frame (length 32)   */
                                /*    ============================   */

                                /* read byte counter registers to    */
                                /* get the number of currently       */
                                /* received bytes                    */
    RecCnt = (WORD) inp (ISAC_R_RBCL) |
                    (WORD) (inp (ISAC_R_RBCH) & 0x0F) < 8;

    if (RecCnt && !(RecCnt &= 0x1F))
       RecCnt = 32;
}
                                /* 'RecCnt' now contains the number */
                                /* of bytes actually received       */

                                /*   already working in receive     */
                                /*   direction?                     */
                                /*   (the flag STATE_REC_ACTIVE is  */
                                /*   set after the first RPF or RME */
                                /*   is detected)                   */
if (!(pt->pt_state & STATE_REC_ACTIVE))
{
   if (RecCnt 0)
   {
      if (rpf)
          pt->pt_rx_curr = pt->pt_rx_start = MMU_req (266);
      else
          pt->pt_rx_curr = pt->pt_rx_start = MMU_req (38);

      if (pt->pt_rx_start == NULL_PTR)
      {
          MMU_ERROR (pei);
          pt->pt_rx_frame = FRAME_NO_MEMORY;
      }
   }

   pt->pt_state|= STATE_REC_ACTIVE;
   pt->pt_rx_cnt = RecCnt;
}
else
                            /* if data has been already          */
                            /*   received only the receive byte */
```

```
                                /*   counter must be updated        */
        pt→pt_rx_cnt += RecCnt;


                                /*   auto-mode, frame greater than
*/
                                /*   260 byte and auto-mode link?
*/
    if (pt→pt_op_mode == MODE_AUTO && pt→pt_rx_cnt 260 &&
        ((inp (ISAC_R_RSTA) & 0x0D) ==9))
    {
        pt→pt_rx_frame = FRAME_OVERFLOW;

                                /*   reset the receiver if incoming */
                                /*   frame exceeds 528 byte I field */
                                /*   length -                       */
                                /*   unbounded frame                */
        if (rpf && pt→pt_rx_cnt 528)
        {
                outp (ISAC_R_CMDR, CMDR_RHR);

                MMU_free (pt→pt_rx_start);

                pt→pt_rx_start  = NULL_PTR;
                pt→pt_state &   = ~STATE_REC_ACTIVE;
                pt→pt_rx_frame  = 0x00;
                pt→pt_rx_cnt    = 0;

                N201_ERROR (pei);
                return;
        }
    }
    else
        if (pt→pt_rx_cnt 266)
                pt→pt_rx_frame = FRAME_OVERFLOW;

                                /* read the bytes from the RFIFO    */
                                /* if no frame error was detected   */
                                /*   and update buffer pointer      */
    if (pt→pt_rx_frame FRAME_ERROR)
    {
       if (RecCnt)
       {
                STRING_IN (pt→pt_rx_curr, ISAC_R_FIFO, RecCnt);
                pt→pt_rx_curr += RecCnt;
       }
    }
```

```
  if (rpf)                          /* return when it was a RPF int.    */
  {
       outp (ISAC_R_CMDR, CMDR_RMD | (inp (ISAC_R_STAR) & CMDR_RNR));
       return;
  }

                                    /*   RME interrupt handling!!!      */
                                    /*   ========================       */

                                    /* the receive status byte is in    */
                                    /*   register RSTA                  */
  rsta = inp (ISAC_R_RSTA);


/*********************************************************************/
  /*   It follows a scanning section to get some information         */
  /*   about the                                                     */
  /*   received frame:                                               */
  /*   - Performed address recognition                               */
  /*   - SAPI ('sapi'), TEI ('tei') and control field                */
  /*     byte(s) ('ctrl')                                            */
  /*     as well as the type of frame (HDLC U, UI, S or I frame) are*/
  /*     determined.                                                 */
  /*   In addition the length of a frame is checked.                 */
/*********************************************************************/


                                    /* set 'pei' according to performed */
                                    /* address recognition              */
  pei       |= ((rsta & 0x0C) > 1) | (rsta & 0x01);
  AutoM     = FALSE;
  tei       = 0;
  sapi      = rsta & 0x02;        /* get the C/R bit value            */
  ptr       = pt->pt_rx_start;
                                    /* depending on the HDLC controller */
                                    /*   operating mode SAPI, TEI and   */
                                    /*   control field information is   */
                                    /*   read                           */
  switch (pt->pt_op_mode)
  {
       case MODE_EXT_TRANSP:        /* no address recognition, SAPI     */
                                    /* and TEI are the first two data */
                                    /*   bytes                          */
            if (pt->pt_rx_cnt 0)
                  pt->pt_rx_cnt--;

            sapi = *ptr++;
```

```
      case MODE_TRANSP:          /* high byte address recognition,   */
                                 /* TEI is the first byte read       */
           if (pt→pt_rx_cnt 2)
                 frame_status = MUTILATED;
           else
                 pt→pt_rx_cnt -= 2;

           tei =*ptr++;
           ctrl = (WORD) *ptr++;

           if (pt→pt_op_mode == MODE_TRANSP)
                 pei |= 0x20;
           else
                 pei |= 0x30;

           break;

      case MODE_AUTO:            /* full address recognition in      */
      case MODE_NON_AUTO:        /* AUTO/nonauto-mode read only the
*/
                                 /* HDLC control field information   */


           if (pt→pt_op_mode == MODE_AUTO)
                                 /* auto-mode link???
*/
                 AutoM = ((rsta & 0x0D) == 0x09) ? TRUE : FALSE,

                 if (!AutoM)
                       pei |= 0x10;

                                 /* the (first byte of the) control*/
                                 /* field is in register RHCR        */
                 ctrl = (WORD) inp (ISAC_R_RHCR);
                 break;
   }

                                 /*  the frame type can be determined*/
                                 /*  using the control field         */
                                 /*  information (least sig. bits)   */
   switch (ctrl & 0x03)
   {
       case 0x3:                 /* *** HDLC U frame**               */
            Two = FALSE;         /* one byte control field!          */

            if (pt→pt_rx_cnt == 0)
```

```
        {
            pt→pt_rx_frame = FRAME_U;
            break;
        }
        else
            pt→pt_rx_frame = FRAME_UI;
                                /*  U frames with I field are always*/
                                /*  transferred up to a protocol    */
                                /*  software entity (regardless     */
                                /*  whether they are valid or not)  */
        break;

case 0x1:                       /* *** HDLC S-Frame**               */

                                /* two byte control field ?         */
        if ((Two = (pt→pt_state & STATE_M128)))
        {
            ctrl <= 8;
            ctrl   |= (WORD) * ptr++;

            if (pt→pt_rx_cnt 0)
                    pt→pt_rx_cnt--;
            else                /* Second byte of the two byte      */
                                /* control field is missing!        */
                    frame_status = MUTILATED;
        }

        if (pt→pt_rx_cnt 0)         /* S frame with I-field!        */
            frame_status = TOO_LONG;

        pt→pt_rx_frame = FRAME_S;
        break;

case 0x2:                       /* *** HDLC I frame**               */
case 0x0:
        Two = (pt→pt_state & STATE_M128);
        pt→pt_rx_frame = FRAME_I;

        if (AutoM)
            break;

        if (Two)                /* two byte control field?          */
        {
            if (pt→pt_rx_cnt == 0)
                    frame_status = MUTILATED;
```

```
                if (pt→pt_rx_cnt 0)
                        pt→pt_rx_cnt- -;

                ctrl <= 8;
                ctrl |= (WORD) *ptr++;
            }
            break;
    }
                                /*   I part greater than 260?       */
    if (pt→pt_rx_cnt 260)       /*   -------------------------      */
    {
        pt→pt_rx_frame = FRAME_OVERFLOW;
                                /*   must reset the controller      */
        outp (ISAC_R_CMDR, CMDR_RMC|CMDR_RHR|CMDR_XRES);
        outp (ISAC_R_TIMR, inp(ISAC_R_TIMR));
        pt→pt_state|= STATE_HDLC_RESET;
        FREE_TX_PATH (PEI);
        N201_ERROR(pei);
    }
    else                        /* enter 'RMC' command if not       */
        outp (ISAC_R_CMDR, CMDR_RMC|(inp (ISAC_R_STAR) & CMDR_RNR));


/*********************************************************************/
    /*                                                               */
    /*    Now all information about the received frame is available: */
    /*        - performed address recognition or TEI and SAPI values.*/
    /*        - HDLC control field                                   */
    /*        - type of frame (HDLC U, UI, S, I frame).              */
    /*        - info about the validity of the frame                 */
    /*                                                               */
/*********************************************************************/

    if (rsta = (rsta & (RSTA_RDO|RSTA_CRC|RSTA_RAB)) ^RSTA_CRC)
            pt→pt_rx_frame = FRAME_FAULT;

    switch (pt→pt_rx_frame)
    {
            case FRAME_FAULT:
                  if (rsta & RSTA_RDO)
                            REC_DATA_OVERFLOW (pei);

                  if (rsta & RSTA_RAB)
                            REC_ABORTED (pei);
```

```
        if (!(rsta & RSTA_CRC))
                CRC_ERROR (pei);

        break;

    case FRAME_S:               /*   HDLC S frame                    */
                                /*   ============                    */
        Decode_S_Frame_BASIC (pei, sapi, tei, ctrl, frame_status,
                        ((pt→pt_state & STATE_M128) ? 0x01 : 0x00));
        MMU_free (pt→pt_rx_start);
        break;

    case FRAME_U:               /*   HDLC U frame                    */
                                /*   ============                    */
        Decode_U_Frame_BASIC (pei, sapi, tei, (BYTE) ctrl);
        MMU_free (pt→pt_rx_start);
        break;

    case FRAME_UI:              /*   HDLC UI or I frame              */
    case FRAME_I:               /*   ==================              */

        if (pt→pt_rx_frame FRAME_ERROR)
        {
               FRAME_PASS          fp;

               fp.mmu_buff         = pt→pt_rx_start;
               fp.start_of_i_data  = ptr;
               fp.i.data_cnt       = pt→pt_rx_cnt;
               fp.Two_byte_cf      = Two;
               fp.ctrl_field       = ctrl;
               fp.pei              = pei;
               fp.frame            = pt→pt_rx_frame | frame_status;
               fp.sapi             = sapi;
               fp.tei              = tei;

                                   /*   transfer the frame to the 'long*/
                                   /*   frame queue'                    */
               PassLongFrame_BASIC (&fp);
        }
        break;

}   /* end of 'switch (pt→pt_rx_frame)'--------------------        */

                                   /* release the data buffer if the   */
```

```
                                /*   frame reception or the frame   */
                                /*   were erroneous                 */
    if (pt→pt_rx_frame = FRAME_ERROR)
          MMU_free (pt→pt_rx_start);

    pt→pt_rx_start  = NULL_PTR;
    pt→pt_state     & = ~STATE_REC_ACTIVE;
    pt→pt_rx_frame  = 0x00;
    pt→pt_rx_cnt    = 0;
}
/*****************************************************************/
/*                                                               */
/*        Function   :Check_TREC_status_ICC ()                   */
/*        Parms      :                                           */
/*        purpose    :called periodically if timer recovery      */
/*                           status was                          */
/*                           detected during previous XPR interrupt */
/*                           handling.                           */
/*                           A transmit-acknowledge for I frame is */
/*                           generated if the                    */
/*                           TREC status is left.                */
/*                                                               */
/*****************************************************************/
EXPORT void
Check_TREC_status_ICC ()
{
    outp (ISAC_R_MASK,
    ~MASK_TIN);                 /* allow only TIN interrupts      */

                                /*   timer recovery status left?  */
    if (!(inp(ISAC_R_STAR2) & STAR2_TREC))
    {
         if (inp(ISAC_R_ISTA)
         & ISTA_TIN)            /* TIN interrupt???               */
         {
              ResetHDLC_ICC ();
              TIN_ERROR (PEI);
         }
         else
                                /*   generate a transmit acknowledge*/
                                /*   I frame if there was no TIN  */
                                /*   interrupt                    */
              TX_ACKNOWLEDGE (PEI, (BYTE) FRAME_I);
         DISABLE_TREC_STATUS_CHECK ();
    }

    outp (ISAC_R_MASK, 0x00);
}
```

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/*   Copyright ©Siemens AG, 1991. All rights reserved.           */
/*   ================================================            */
/*                                                               */
/*   File      :   sbc.c                                         */
/*   Function  :   ISAC-S L1 driver routines                     */
/*   Created   :   Jan. 1991                                     */
/*   Compiler  :   Microsoft(R) C Optimizing Compiler, version 5.1 */
/*                                                               */
/*****************************************************************/

     /*   Include Files                                          */
     /*   =============                                          */

#include "def.h"                /*   type redefinitions         */

#include "icc.h"                /*   ICC / ISAC-S specific      */
                                /*   definitions                */

#include "message.h"            /*   mascros for status and error */
                                /*   messages                   */


          /*   CI codes                                         */
          /*****************************************************/

#define  CI_PU     (BYTE)0x1C    /*   0111 PU indication        */
#define  CI_TIM    (BYTE)0x00    /*   0000 timing requested     */
#define  CI_AI     (BYTE)0x30    /*   1100 activation indication */
#define  CI_AR     (BYTE)0x20    /*   1000 activation request   */
#define  CI_DIU    (BYTE)0x3C    /*   1111 deactivation ind. upstream*/
#define  CI_DID    (BYTE)0x3C    /*   1111 deactivation ind. downst. */
#define  CI_DR     (BYTE)0x00    /*   0000 deactivation request */
#define  CI_RS     (BYTE)0x04    /*   0001 Reset                */
#define  CI_EI     (BYTE)0x18    /*   0110 Error indicate downstream */


    /*   Imported Functions                                     */
    /*   ==================                                     */

        /* from   crt0.asm                                      */
IMPORT   unsigned int ENTERNOINT ();
IMPORT   void              LEAVENOINT ();
```

```
    /* Export Functions                                                */
    /*   ================                                              */

EXPORT   int              InitL1_SBC (void);
EXPORT   int              ActL1_SBC (void);
EXPORT   int              ArlL1_SBC (void);
EXPORT   void             IntL1_SBC (void);

EXPORT   int              ResL1_SBC (void);
EXPORT   int              EnaClk_SBC (void);


    /*   Variables                                                     */
    /*   =========                                                     */

LOCAL         BYTE    CI_received;


    /*   Function Declaration                                          */
    /*   ====================                                          */

/*********************************************************************/
/*                                                                   */
/*       Function  :EnaClk_SBC ()                                    */
/*       Parms     :none                                             */
/*       purpose   :enable clocks for TE configurations.             */
/*                                                                   */
/*********************************************************************/

EXPORT int
EnaClk_SBC ()
{
    unsigned int count, i = 0;
    BYTE              BitSet, spcr;
                                /* Test to see if clocks are        */
                                /* actually there. Because the SBC  */
                                /* after reset does not deactivate  */
                                /* its clocks immediately we will   */
                                /* make pretty sure that the clocks */
                                /*   are there before we leave this */
                                /*   routine                        */
    BitSet = inp (ISAC_R_STAR) & STAR_BVS;
    count = 0;
                                /* we test to see if 6 changes in   */
```

```
                                /* the STAR:BVS bit indicating the  */
                                /* reception of at least 3 frames   */
                                /* (6 B channels). If at any time   */
                                /* we fail to find a bit change     */
                                /* and the counter i reaches its    */
                                /* maximum then we assume that      */
                                /* clocks are no longer present     */
    for (i = 0; i 500; i++)
        if ((inp(ISAC_R_STAR) & STAR_BVS) != BitSet)
        {                       /* Of course we have to reset our   */
                                /* counter every time a bit change  */
            if (++count 6)      /* is observed to give the next     */
                return (FALSE); /* bit change the same amount of    */
                                /* time in which to occur!!!        */
            i = 0;
            BitSet = inp (ISAC_R_STAR) & STAR_BVS;
        }

                                /* the Bx versions require one edge */
                                /*    at FSC.                       */
                                /* Otherwise the setting of the SPU */
                                /* has no effect (result: no clock) */
                                /* The IOM direction control bit    */
                                /* IDC in the SQXR register         */
                                /* is set before and reset after    */
                                /* the system is clocking           */
    outp (ISAC_R_SQXR, 0x80);

    spcr = inp(ISAC_R_SPCR);
    outp (ISAC_R_SPCR, spcr|SPCR_SPU);

    outp (ISAC_R_CIX0, CIXR_TBC|CI_TIM|0x03);
                                /* wait for power up indication     */
    while ((inp(ISAC_R_CIX0) & CIR_MASK) != CI_PU)
        if (++i 1000)
            break;              /* time out                         */

    outp (ISAC_R_SPCR, spcr);   /* restore SPCR                     */
    outp (ISAC_R_SQXR, 0x00);   /* now reset the IDC bit            */

    return (TRUE);
}
```

```
/*****************************************************************/
/*                                                               */
/*        Function    :InitL1_SBC ()                             */
/*        Parms       :none                                      */
/*        purpose     :initialize L1 part of ISAC-S              */
/*                     reset L1 to come to a default state       */
/*                                                               */
/*****************************************************************/
EXPORT int
InitL1_SBC ()
{
    BYTE                r_mode;

    outp (ISAC_R_MASK, 0xFF);
    r_mode = inp (ISAC_R_MODE);

                                /* timing mode 0 is used         */
    outp (ISAC_R_MODE, (r_mode & ~(MODE_HMD2|MODE_HMD1)) |
          MODE_HMD0);
    outp (ISAC_R_ADF2, 0x80);   /* program IOM2 mode in ICC/ISAC-S */
    outp (ISAC_R_SPCR, 0x00);   /* terminal mode                 */
    outp (ISAC_R_STCR, 0x70);   /* TIC bus address '7'           */
                                /* no watchdog timer             */

    outp (ISAC_R_MASK, 0x00);

    if (!ResL1_SBC ())
        return (ACK_ACCESS_FAULT);

    return (ACK_DONE);
}
/*****************************************************************/
/*                                                               */
/*        Function    :ActL1_SBC ()                              */
/*        Parms       :none                                      */
/*        purpose     :establish L1 link (= activation)          */
/*                                                               */
/*****************************************************************/
EXPORT int
ActL1_SBC ()
{
                                /* the activation procedure is not */
                                /* initiated if the layer-1 link is */
                                /* already established. In that  */
                                /* case only an activation       */
                                /* indication message is generated */
    if (((CI_received = inp(ISAC_R_CIX0)) & CIR_MASK) != CI_AI)
    {
```

```
            EnaClk_SBC ();

                outp (ISAC_R_CIX0, CIXR_TBC|CI_AR|0x03);

            return (ACK_DONE);
        }

        DECODE_L1_STATUS (PEI, CI_received);
        return (ACK_DONE);
}
/*****************************************************************/
/*                                                               */
/*        Function    :ArlL1_SBC ()                              */
/*        Parms       :none                                      */
/*        purpose     :activate local loop                       */
/*                                                               */
/*****************************************************************/

EXPORT int
ArlL1_SBC ()
{
    EnaClk_SBC ();

    outp (ISAC_R_CIX0, 0x6B);

    return (ACK_DONE);
}
/*****************************************************************/
/*                                                               */
/*        Function    :IntL1_SBC ()                              */
/*        Parms       :none                                      */
/*        purpose     :handle C/I interrupts                     */
/*                                                               */
/*****************************************************************/

EXPORT void
IntL1_SBC ()
{
    CI_received = inp           /* read CIRR (CIR0) register      */
    (ISAC_R_CIX0);

                                /* power down SBC/ISAC-S if it has  */
                                /* changed from activated to        */
                                /* pending mode                     */
    if ((CI_received & CIR_MASK) == CI_DR)
        outp (ISAC_R_CIX0, CIXR_TBC|CI_DIU|0x03);

                                /* activation confirmation in IOM2  */
```

```
                                /* configurations. The SBC          */
                                /* (ISAC-S) must confirm an         */
                                /* activation from network side.    */
                                /* Only then it will be transparent */
                                /*   for upstream B channel data    */
    if ((CI_received & CIR_MASK) == CI_AI)
        outp (ISAC_R_CIX0, CIXR_TBC|CI_AR|0x03);

    DECODE_L1_STATUS (PEI, CI_received);
}
/*****************************************************************/
/*                                                               */
/*        Function    :ResL1_SBC ()                              */
/*        Parms       :none                                      */
/*        purpose     :Reset SBC / L1 part of ISAC-S             */
/*                     (also used for device test)               */
/*                                                               */
/*****************************************************************/

EXPORT int
ResL1_SBC ()
{
    int       i, state, failed = FALSE;
    BYTE      Loop, r_spcr;

    state = ENTERNOINT ();      /* disable all system interrupts    */

                                /* if testloop mode was programmed  */
                                /* switch it off to enable L1       */
                                /* status recognition               */
    r_spcr = inp (ISAC_R_SPCR);

    if (Loop = (r_spcr & SPCR_TPL))
        outp (ISAC_R_SPCR, (r_spcr & ~SPCR_TPL));

    outp (ISAC_R_MASK,          /* allow only C/I interrupts        */
    ~ISTA_CIC);

    EnaClk_SBC ();

                                /* output the command code          */
    outp (ISAC_R_CIX0, (BYTE) (CIXR_TBC|CI_RS|0x03));

    i = 0;
                                /* wait for the expected state      */
    while ((inp(ISAC_R_CIX0) & CIR_MASK) != CI_EI)
        if (i++ 20000)
```

```
        {                           /*   break if timeout                */
            failed = TRUE;
            break;
        }

                                    /*   output the release command      */
    outp (ISAC_R_CIX0, (BYTE) (CIXR_TBC | CI_DIU | 0x03));

                                    /* Wait for DIU or AIU because       */
                                    /* it can cause problems for the     */
                                    /* enable clock routine if the       */
                                    /* clocks disappear mid routine      */
                                    /* due to an earlier reset           */

    for (i = 0; i 20000; i++)
    {
        CI_received = inp (ISAC_R_CIX0) & CIR_MASK;

        if ((CI_received == CI_DIU) || (CI_received == CI_AI))
            break;
    }

    if (CI_received == CI_AI)
        outp (ISAC_R_CIX0, CIXR_TBC|CI_AR|0x03);

    if (Loop)                       /* restore original value of SPCR    */
        outp (ISAC_R_SPCR, r_spcr);

    outp (ISAC_R_MASK, 0x00);       /* enable interrupts again           */
    LEAVENOINT (state);

    return (TRUE);
}
```

```
/*****************************************************************************/
/*                                                                           */
/*   Copyright ©Siemens AG, 1991. All rights reserved.                       */
/*   ================================================                        */
/*                                                                           */
/*   File       :  icc.h                                                     */
/*   Function   :  definitions for the ICC / ISAC-S driver routines          */
/*   Created    :  Jan. 1991                                                 */
/*   Compiler   :  Microsoft(R) C Optimizing Compiler, version 5.1           */
/*                                                                           */
/*****************************************************************************/


     /*   Register addresses                                                 */
     /*   ==================                                                 */
                              /*   In the SIPB environment                  */
                                   all ISDN
                              /*   devices are addressed in the IO*/
                              /*   space. But the IO space is               */
                              /*   mapped into the memory (80188            */
                              /*   feature). That results in four           */
                              /*   byte 'long pointers' to address*/
                              /*   ISDN devices' registers.                 */

#define BASE          ((IO_PORT)0x60000080L)


                              /*   please find below the defini-            */
                              /*   tions for the register                   */
                                   addresses
                              /*   and the corresponding register           */
                              /*   mnemonic in comments                     */

#define ISAC_R_FIFO (BASE+0x00) /* RFIFO / XFIFO                            */
#define ISAC_R_MASK (BASE+0x20) /* MASK                                     */
#define ISAC_R_ISTA ISAC_R_MASK /* ISTA                                     */
#define ISAC_R_CMDR (BASE+0x21) /* CMDR                                     */
#define ISAC_R_STAR ISAC_R_CMDR /* STAR                                     */
#define ISAC_R_MODE (BASE+0x22) /* MODE                                     */
#define ISAC_R_TIMR (BASE+0x23) /* TIMR                                     */
#define ISAC_R_XAD1 (BASE+0x24) /* XAD1                                     */
#define ISAC_R_EXIR ISAC_R_XAD1 /* EXIR                                     */
#define ISAC_R_XAD2 (BASE+0x25) /* XAD2                                     */
#define ISAC_R_RBCL ISAC_R_XAD2 /* RBCL                                     */
#define ISAC_R_SAP1 (BASE+0x26) /* SAP1                                     */
#define ISAC_R_SAP2 (BASE+0x27) /* SAP2                                     */
#define ISAC_R_RSTA ISAC_R_SAP2 /* RSTA                                     */
```

```
#define ISAC_R_TEI1 (BASE+0x28) /* TEI1                                  */
#define ISAC_R_TEI2 (BASE+0x29) /* TEI2                                  */
#define ISAC_R_RHCR ISAC_R_TEI2 /* RHCR                                  */
#define ISAC_R_SPCR (BASE+0x30) /* SPCR                                  */
#define ISAC_R_CIX0 (BASE+0x31) /* CIX0 = CIR0                           */
#define ISAC_R_MONR (BASE+0x32) /* MONR                                  */
#define ISAC_R_SSGX (BASE+0x33) /* SSGX                                  */
#define ISAC_R_SFCR (BASE+0x34) /* SFCR                                  */
#define ISAC_R_STCR (BASE+0x37) /* STCR                                  */
#define ISAC_R_ADFR (BASE+0x38) /* ADFR                                  */

#define ISAC_R_STAR2(BASE+0x2B) /* STAR2                                 */
#define ISAC_R_RBCH (BASE+0x2A) /* RBCH                                  */
#define ISAC_R_CIX1 (BASE+0x33) /* CIX1 = CIR1                           */
#define ISAC_R_MOX1 (BASE+0x34) /* MOX1 = MOR1                           */
#define ISAC_R_ADF2 (BASE+0x39) /* ADF2                                  */
#define ISAC_R_MOCR (BASE+0x3A) /* MOCR = MOSR                           */
#define ISAC_R_SQXR (BASE+0x3B)                                         */


                    /*   register-flags                                 */
                    /*   --------------                                 */

                    /* ISTA (interrupt status register)                 */

#define  ISTA_RME   (BYTE)0x80  /*   Receive Message End                */
#define  ISTA_RPF   (BYTE)0x40  /* Receive Pool Full                    */
#define  ISTA_RSC   (BYTE)0x20  /* Receive Status Change                */
                                /*   (used in auto-mode only)           */
#define  ISTA_XPR   (BYTE)0x10  /*   Transmit Pool Ready                */
#define  ISTA_TIN   (BYTE)0x08  /*   Timer Interrupt                    */

                                /*   for ISAC-S and ICC:                */
#define  ISTA_CIC   (BYTE)0x04  /*   C/I Code Change                    */
#define  ISTA_SIN   (BYTE)0x02  /*   Synchronous Transfer Register      */
#define  ISTA_EXI   (BYTE)0x01  /*   Extended Interrupt                 */


                    MASK (interrupt mask register)                      */

#define  MASK_RME   (BYTE)0x80
#define  MASK_RPF   (BYTE)0x40
#define  MASK_RSC   (BYTE)0x20
#define  MASK_XPR   (BYTE)0x10
#define  MASK_TIN   (BYTE)0x08
```

```
#define  MASK_CIC    (BYTE)0x04
#define  MASK_SIN    (BYTE)0x02
#define  MASK_EXI    (BYTE)0x01


                     /* EXIR (extended interrupt register)          */

#define  EXIR_XMR    (BYTE)0x80  /* Transmit Message Repeat          */
#define  EXIR_XDU    (BYTE)0x40  /*   Transmit Data Underrun         */
#define  EXIR_PCE    (BYTE)0x20  /*   Protocol Error                 */
#define  EXIR_RFO    (BYTE)0x10  /*   Receive Frame Overflow         */

                                 /* ICC/ISAC:                        */
#define  EXIR_SOV    (BYTE)0x08  /*   Synchronous Transfer Overflow  */
#define  EXIR_MOR    (BYTE)0x04  /* MON channel status change        */
#define  EXIR_SAW    (BYTE)0x02  /* Subscriber Awake                 */
#define  EXIR_WOV    (BYTE)0x01  /* Watchdog Timer Overflow          */


                     /* STAR (status register)                       */
#define  STAR_BVS    (BYTE)0x02  /* B-channel Valid at SLD (only     */
                                 /*   ICC/ISAC)                      */
#define  STAR_XFW    (BYTE)0x40  /* transmit FIFO write enable, data */
                                 /*   can be written into the XFIFO  */


                         /* STAR2 (status register #2, only in      */
                         /* ICC B4, ISAC-S B3 or                    */
                         /*               later versions)           */

#define  STAR2_TREC  (BYTE)0x02  /* timer recovery status            */
#define  STAR2_WFA   (BYTE)0x08  /* waiting for acknowledge          */


                     /*RSTA (receive status register)              */

#define  RSTA_RDO    (BYTE)0x40  /*   Receive Data Overflow          */
#define  RSTA_CRC    (BYTE)0x20  /* CRC compare/check                */
#define  RSTA_RAB    (BYTE)0x10  /* Receive Message Aborted          */


                     /* CMDR (command register)                     */

#define  CMDR_RMC        (BYTE)0x80   /* Receive Message Complete    */
#define  CMDR_RHR        (BYTE)0x40   /* Reset HDLC Receiver         */
```

```
#define  CMDR_RNR    (BYTE)0x20  /* Receiver Not Ready -                */
                                 /*    (used in auto-mode only)        */
#define  CMDR_STI    (BYTE)0x10  /* Start Timer                        */
#define  CMDR_XTF    (BYTE)0x08  /* Transmit Transparent Frame         */
#define  CMDR_XIF    (BYTE)0x04  /* Transmit I-Frame (used in auto     */
                                 /*    mode only)                      */
#define  CMDR_XME    (BYTE)0x02  /* Transmit Message End               */
#define  CMDR_XRES   (BYTE)0x01  /* Transmit Reset                     */


                     /* MODE (Mode Register)                          */

#define  MODE_MDS1   (BYTE)0x80  /* Mode Select 1          (MDS2)      */
#define  MODE_MDS0   (BYTE)0x40  /* Mode Select 2          (MDS1)      */
#define  MODE_ADM    (BYTE)0x20  /* Address Mode           (MDS0)      */
#define  MODE_TMD    (BYTE)0x10  /* Timer Mode                         */
#define  MODE_RAC    (BYTE)0x08  /* Receiver Active                    */

                                 /*    ICC (ISAC) serial port b mode:  */
#define  MODE_HMD2   (BYTE)0x04  /* HDLC port Mode2                    */
#define  MODE-HMD1   (BYTE)0x02  /* HDLC port Mode1                    */
#define  MODE-HMD0   (BYTE)0x01  /* HDLC port Mode0                    */


                     /* CIXR   Register  (CIX0 when ICC-B (ISAC)      */

#define  CIXR_TBC    (BYTE)0x40  /* TIC bus Control                    */
#define  CIXR_TCX    (BYTE)0x02  /* T-channel transmit                 */
#define  CIXR_ECX    (BYTE)0x01  /* E-channel transmit                 */


                     /* STCR (ICC/ISAC) (synchronous transfer         */
                     /* control register)                             */

#define  STCR_ST1    (BYTE)0x08  /*    Enable SIN interrupt at the     */
                                 /*    beginning of 8 kHz frame signal*/
#define  STCR_ST0    (BYTE)0x04  /*    Enable SIN interrupt at center  */
                                 /*    of 8 kHz frame signal           */
#define  STCR_SC1    (BYTE)0x02  /*    Synchronous Transfer 1 complete*/
#define  STCR_SC0    (BYTE)0x01  /*    Synchronous Transfer 0 complete*/


                     /* SPCR (ICC/ISAC) (serial port                  */
                     /* control register)                             */

#define  SPCR_SPU    (BYTE)0x80  /* Software Power Up - (in TE mode   */
```

```
                                        /*   only)                          */
#define  SPCR_SAC    (BYTE)0x40   /* SIP Activated                        */
#define  SPCR_SPM    (BYTE)0x20   /* Serial Port Timing Mode              */
#define  SPCR_TPL    (BYTE)0x10   /* Test Loop                            */
#define  SPCR_B1C1   (BYTE)0x08   /* Switching of B1 channel    (IOM1)    */
#define  SPCR_B1C0   (BYTE)0x04"  /*    "                       (IOM1)    */
#define  SPCR_B2C1   (BYTE)0x02   /* Switching of B2 channel    (IOM1)    */
#define  SPCR_B2C0   (BYTE)0x01"  /*    "                       (IOM1)    */


#define  MONX_ID     (BYTE)0x80   /* Monitor Channel ID command           */

                                        /*   mask for the CIRR / CIR0       */
#define  CIR_MASK    (BYTE)0x3C   /*   registers to gain the CI code      */


                                        /* values of 'pt_op_mode'           */
                                        /* (stores operation mode of HDLC   */
                                        /*                  controller)     */

#define  MODE_AUTO           0    /* -Auto-Mode                           */
#define  MODE_NON_AUTO       1    /* -non-auto-mode                       */
#define  MODE_TRANSP         2    /* -transparent mode                    */
                                        /*   (high byte address recognition)*/
#define  MODE_EXT_TRANSP     3    /* -extended transparent mode           */
                                        /*   (no address recognition)       */
#define  MODE_NOT_FIXED      0xFF /* -mode not fixed                      */


                                        /* values of the frame identifiers  */
                                        /* 'pt_rx_frame' and 'pt_tx_frame'  */
                                        /* (store frame type + errors)      */
#define  FRAME_I     (BYTE)  0    /* I-frame                              */
#define  FRAME_UI    (BYTE)  1    /* UI-frame                             */
#define  FRAME_TR    (BYTE)  2    /* transparent data                     */
#define  FRAME_S     (BYTE)  4    /* S-frame                              */
#define  FRAME_U     (BYTE)  8    /* U-frame                              */

#define  VALID       (BYTE)  0x00
#define  MUTILATED   (BYTE)  0x80
#define  TOO_LONG    (BYTE)  0x82

#define  FRAME_ERROR         100  /* indicates an error (in general)      */
#define  FRAME_OVERFLOW      100  /* indicates frame overflow             */
#define  FRAME_NO_MEMORY101       /* no memory was free to store the      */
```

```
                                /*   frame
#define FRAME_FAULT  102        /* frame is erroneous:                */
                                /*   - CRC failed                     */
                                /*   - RDO receive data overflow      */
                                /*   - RAB receiver aborted           */

                                /* flags of 'pt_state'                */
                                /* 'pt_state' contains flags          */
                                /*   representing the various states*/
                                /*   of the ISDN device.              */
                                /*   ---------------------   */
#define STATE_TX_MMU_FREE 0x0001/* memory buffer must be released   */
                                /* after transmission                 */
#define STATE_TX_ACTIVE   0x0002/* device (and driver software) is  */
                                /*   operating in transmit direction*/
#define STATE_REC_ACTIVE  0x0004/* operating in receive direction   */
#define STATE_HDLC_RESET  0x0008/* HDLC controller was reset via    */
                                /* the command                        */
#define STATE_LOOP        0x0010/* controller in test loop mode     */
#define STATE_M128        0x0100/* modulo 128 (extended) operation  */
#define STATE_L1_CTRL     0x0200/* device is used to program L1     */
#define STATE_IOM2        0x1000/* device operates in IOM2          */
                                /*   configuration                    */

                                /*   the following flag is only     */
                                /*   important in auto-mode.        */
                                /*   It is set                      */
                                /*   if an UI frame (ID check       */
                                /*   response is sent while the ICC/*/
                                /*   ISAC-S is waiting for          */
                                /*   acknowledge for a previously   */
                                /*   sent HDLC I frame              */
#define UI_SENT_WHILE_                  0x2000
        WAITING_FOR_ACK



    /* Structures
    /*    ===========
                                /* the following structure has been */
                                /*   defined to hold all device
                                     specific
                                /*   information
struct isac
{
unsigned int    pt_op_mode,          /* operation mode of the HDLC */
                                     /* controller (auto-mode, ..) */
```

```
            pt_state;           /* device status                        */


    FPTR        pt_tx_start;    /* tx start pointer                     */
    FPTR        pt_tx_curr;     /* tx current pointer                   */
    FPTR        pt_rx_start;    /* rx start pointer                     */
    FPTR        pt_rx_curr;     /* rx current pointer                   */
    WORD        pt_tx_cnt,      /* transmit byte counter                */
                pt_rx_cnt;      /* receive byte counter                 */
    BYTE        pt_rx_frame,    /* type of received frame               */
                pt_tx_frame,    /* type of sent frame                   */
                pt_CI_rec;      /* received CI code (L1 device          */
                                /*   status)                            */
;
typedef struct isac ISAC;



#define   PEI   (BYTE)0x00      /*   protocol entity identifier.        */
                                /*   It is only necessary for the       */
                                /*   driver test in SIPB environment*/


    /*   Error Codes for Resp_Status_BASIC ()                           */
    /*   (SIPB system specific error message generation)                */
    /*   ==============================================                 */

    /* status byte 3
#define  RESP_ERR_MMU     0x08L  /*   no memory                        */
#define  RESP_ERR_N201    0x10L  /* frame to long                      */
#define  RESP_ERR_READY   0x20L  /* Peer Received Ready                */
#define  RESP_ERR_BUSY    0x21L  /* Peer Receiver Busy                 */

#define  RESP_ERR_TIN     0x40L  /* TIMER interrupt                    */

              /* status byte 2                                         */
                                /* Receive Data Overflow                */
#define  RESP_ERR_RDO (DWORD) RSTA_RDO < 8L
                                /* CRC-Error                            */
#defineRESP_ERR_CRC (DWORD) RSTA_CRC <8L
                                /* Receive Frame aborted                */
#defineRESP_ERR_RAB (DWORD) RSTA_RAB <8L
```

```
                /* status byte 1                                          */
                                        /* Message Lost/Transmission Error*/
#define   RESP_ERR_XMR  (DWORD) EXIR_XMR < 15L
                                        /* Message Lost/Data Underrun     */
#define   RESP_ERR_XDU  (DWORD) EXIR_XDU < 15L
                                        /* N(R)-Error + Unexpected I-Data */
#define   RESP_ERR_PCE  (DWORD) EXIR_PCE < 15L
                                        /* Receive Frame overflow         */
#define   RESP_ERR_RFO  (DWORD) EXIR_RFO < 15L
```