Handbuch Kommunikation

Handbuch Kommunikation

Obwohl bei der Erstellung dieser Dokumentation große Sorgfalt verwendet wurde, kann EPIS Microcomputer nicht für die vollständige Richtigkeit der darin enthaltenen Informationen garantieren und übernimmt keinerlei Verantwortung, weder für darin vorkommende Fehler, noch für eventuell auftretende Schäden, die auf Grund ihrer Verwendung entstehen.

Die beschriebenen Hard- und Softwareprodukte der Fa. EPIS Microcomputer unterliegen einer ständigen Weiterentwicklung bezüglich Funktion, Verwendung und Präsentation. Ihre Beschreibung hat daher keinerlei verbindlichen, vertragsgemäßen Charakter.

Die in diesem Handbuch wiedergegebenen Angaben gelten uneingeschränkt nur für die jeweils aktuelle Version des EMD-Betriebssystems.

Der aktuelle Stand der einzelnen Kapitel ist in den Inhaltsverzeichnissen vermerkt. Weiterentwicklungen und Änderungen zu älteren Ständen sind zusätzlich als Index gekennzeichnet.

Dabei gilt folgende Symbolik:
+ = Ergänzung zum bisherigen Inhalt
* = Änderung des bisherigen Inhalts
- = nicht mehr gültiger Inhalt wurde entfernt

EPIS Microcomputer GmbH
Rüdiger Schulz Dipl. Ing. (FH)
Lautlinger Str. 147, D-72458 Albstadt

© Copyright EPIS Microcomputer GmbH 2003.

Sicherheitstechnische Hinweise

ACHTUNG:
Dieses Symbol wird überall dort eingesetzt, wo auf die genaue Einhaltung der angegebenen Vorschrift bzw. Werte geachtet werden muß. Bei Nicht-einhaltung besteht die Gefahr der Fehlfunktion bzw. Zerstörung wobei u. U. gleichzeitig Gefahr für Leib und Leben bestehen kann.

HINWEIS:
Mit diesem Symbol sind Vorschläge und Tips für die Verwendung und Programmierung der Geräte versehen. Eine Einhaltung der Vorschläge und Tips wird angeraten.

Diese Anleitung enthält die erforderlichen Informationen für den bestimmungsgemäßen Gebrauch der darin beschriebenen Produkte. Sie wendet sich an technisch qualifiziertes Personal, das

❶ entweder als Projektierungspersonal mit den Sicherheits-Konzepten der Automatisierungstechnik vertraut ist;

❷ oder als Bedienungspersonal im Umfang mit Einrichtungen der Automatisierungstechnik unterwiesen ist und den auf die Bedienung bezogenen Inhalt dieser Anleitung kennt;

❸ oder als Inbetriebsetzungs- und Servicepersonal eine zur Reparatur derartiger Einrichtungen der Automatisierungstechnik befähigende Ausbildung besitzt bzw. die Berechtigung hat, Stromkreise und Geräte/Systeme gemäß des Standards der Sicherheitstechnik in Betrieb zu nehmen, zu erden und zu kennzeichnen.

Die Produkte werden entsprechend den einschlägigen VDE-Bestimmungen, VDE-Vorschriften und IEC-Empfehlungen konstruiert, hergestellt und geprüft.

Gefahrenhinweise:

Diese Hinweise dienen einerseits als Leitfaden für die am Projekt beteiligten Personen und andererseits der Sicherheit vor Beschädigung des beschriebenen Produktes oder angeschlossener Geräte.

Sicherheitstechnische Hinweise

Bestimmungsgemäßer Gebrauch, Geräteaufbau und Montage

Das Gerät darf nur für die im Handbuch und in der technischen Beschreibung vorgesehenen Einsatzfälle und nur in Verbindung mit von EPIS Microcomputer empfohlenen bzw. zugelassenen Fremdgeräten und -Komponenten verwendet werden.

ACHTUNG:
Alle in diesem Handbuch beschriebenen Funktionen werden im vollen Umfang nur bei Verwendung des jeweils neuesten Gerätestandes gewährleistet.

Weiter ist zu beachten, daß

❶ der einwandfreie und sichere Betrieb des Produktes sachgemäßen Transport, sachgerechte Lagerung, Aufstellung und Montage sowie sorgfältige Bedienung voraussetzt.

❷ das Automatisierungsgerät spannungsfrei sein muß, bevor es montiert, demontiert oder der Aufbau verändert wird.

❸ die Systeme nur durch eine Fachkraft installiert werden dürfen. Dabei sind die entsprechenden Vorschriften nach DIN und VDE zu berücksichtigen.

Hinweise zur Projektierung und Installation des Produktes

☞ Die im spezifischen Einsatzfall geltenden Sicherheits- und Unfallverhütungsvorschriften sind zu beachten.

☞ Bei Einrichtungen mit festem Anschluß (ortsfeste Geräte/Systeme) ohne allpoligen Netztrennschalter und/oder Sicherungen ist ein Netztrennschalter oder eine Sicherung in die Gebäude-Installation einzubauen; die Einrichtung ist an einen Schutzleiter anzuschließen

☞ Bei Geräten, die mit Netzspannung betrieben werden, ist vor Inbetriebnahme zu kontrollieren, ob der eingestellte Netzspannungsbereich mit der örtlichen Netzspannung übereinstimmt.

☞ Bei 24V-Versorgung ist auf eine sichere elektrische Trennung der Kleinspannung zu achten. Nur nach IEC 364-4-41 bzw. HD 384.04.41 (VDE 0100 Teil 410) hergestellte Netzgeräte verwenden.

☞ Not-Aus-Einrichtungen gemäß EN 60204/IEC 204 (VDE 0113) müssen in allen Betriebsarten der Automatisierungseinrichtung wirksam bleiben. Entriegeln der Not-Aus-Einrichtungen darf keinen unkontrollierten oder undefinierten Wiederanlauf bewirken.

☞ Einrichtungen der Automatisierungstechnik und deren Bedienelemente sind so einzubauen, daß diese gegen unbeabsichtigte Betätigung ausreichend geschützt sind.

Sicherheitstechnische Hinweise

Verhütung von Material- oder Personenschäden

- ☞ Die unter der Bezeichnung „Grenzwert“ angegebenen Spannungswerte dürfen weder unterschritten noch überschritten werden, da dieses zu Fehlfunktionen bzw. zur Zerstörung der Geräte führen kann.

- ☞ Überall dort, wo in der Automatisierungseinrichtung auftretende Fehler große Materialschäden oder sogar Personenschäden verursachen können, müssen zusätzliche externe Sicherheitsvorkehrungen getroffen oder Einrichtungen geschaffen werden, die auch im Fehlerfall einen definierten Betriebszustand gewährleisten bzw. erzwingen (z.B. durch unabhängige Grenzwertschalter, mechanische Verriegelungen usw.).

Im übrigen verweisen wir auf die “Allgemeinen Geschäftsbedingungen“ der EPIS Microcomputer GmbH.

Sicherheitstechnische Hinweise

Inhaltsübersicht

	Index	Seite
1.1 Inhaltsübersicht		1
2.1 Inhaltsverzeichnis PLC		3
2.2 Allgemeines		3
2.3 STRING-X Datentyp und Befehle		4
2.4 Übersicht der Flags und Definitionen		5
2.4.1 Initialisierungsarten		6
2.4.2 Übertragungsgeschwindigkeiten		7
2.5 Standard-Initialisierung		7
2.6 Auswahl der Schnittstellenparameter		8
2.7 Bearbeitung und Inhalt des Datenfeld DEC		10
2.8 Definition von Start- und Stop-Sequenzen		13
2.9 Bearbeitung und Inhalt des Datenfeld SEQ		14
2.10 Bearbeitung und Inhalt des Datenfeld FLG		15
2.11 Senden und Empfangen von Daten		17
2.12 Fehlererkennung		18
2.13 Kommunikation mit StringX		19
2.13.1 Beschreibung der Funktionen		19
2.13.2 Beispielprogramm String		30
2.14 Serielle Kommunikation		36
2.14.1 Beschreibung der BCom-Funktionen		37
2.14.2 Beispielprogramm		58

Inhaltsübersicht

3.1 Inhaltsverzeichnis C 1

3.2 Allgemeines 3

3.3 STRING-X Datentyp und Befehle 4

3.4 Übersicht der Flags und Definitionen 5

3.5 Bearbeitung und Inhalt des Datenfeld DEC 6

3.6 Definition von Start- und Stop-Sequenzen 9

3.7 Bearbeitung und Inhalt des Datenfeld SEQ 10

3.8 Bearbeitung und Inhalt des Datenfeld FLG 11

3.9 Standard-Initialisierung 14

3.10 Auswahl der Schnittstellenparameter 15

3.11 Senden und Empfangen von Daten 17

3.12 Fehlererkennung 18

3.13 Kommunikation mit StringX 19

3.13.1 Beschreibung der Funktionen 19

3.13.2 Beispielprogramm String 24

3.14 Serielle Kommunikation 25

3.14.1 Allgemeines 25

3.14.2 Bcom Funktionsbeschreibung 26

Inhaltsübersicht

Inhaltsübersicht

Inhaltsverzeichnis

	Index	Seite
2.1	Inhaltsverzeichnis	1
2.2	Allgemeines	3
2.3	STRING-X Datentyp und Befehle	4
2.4	Übersicht der Flags und Definitionen	5
2.4.1	Initialisierungsarten	6
2.4.2	Übertragungsgeschwindigkeiten	7
2.5	Standard-Initialisierung	7
2.6	Auswahl der Schnittstellenparameter	8
2.7	Bearbeitung und Inhalt des Datenfeld DEC	10
2.8	Definition von Start- und Stop-Sequenzen	13
2.9	Bearbeitung und Inhalt des Datenfeld SEQ	14
2.10	Bearbeitung und Inhalt des Datenfeld FLG	15
2.11	Senden und Empfangen von Daten	17
2.12	Fehlererkennung	18
2.13	Kommunikation mit StringX	19
2.13.1	Beschreibung der Funktionen	19
2.13.2	Beispielprogramm String	30
2.14	Serielle Kommunikation	36
2.14.1	Beschreibung der BCom-Funktionen	37
2.14.2	Beispielprogramm	58

Inhaltsverzeichnis

2.2 Allgemeines

Mit der Bibliothek 9010.lib wird eine umfangreiche Programmierschnittstelle zur Verfügung gestellt, die eine Kommunikation zwischen zwei oder mehreren sich in einem Netzwerk befindlichen Teilnehmern möglich macht.

Die in diesem Abschnitt beschriebene Bibliothek „9010.liB“ stellt dem Anwender Funktionen zur Verfügung mit denen ein Protokoll aufgebaut werden kann, die Schnittstelle initialisiert wird und Daten gesendet bzw. empfangen werden können.

HINWEIS:
Während der StringX aktiv ist, kann keine ONLINE-Verbindung über die Programmierschnittstelle mit GRIPS_PLC hergestellt werden. Es empfiehlt sich daher im PLC-Programm eine Möglichkeit zur Deaktivierung von StringX vorzusehen.

In der Bibliothek „9010.lib“ sind zur einfachen Programmierung bereits Namen für die einzelnen Felder vorgegeben. Zur übersichtlichen Programmierung empfiehlt es sich, diese Namen zu verwenden. Die folgende Dokumentation der Bibliothek „9010.lib“ verwendet diese Namen.

2.3 STRING-X Datentyp und Befehle

STRING-X Datentyp

```
TYPE STRX
STRUCT
     RBOX : ARRAY[0..255] OF BYTE; (* Empfangsdaten *)
     TBOX : ARRAY[0..255] OF BYTE; (* Sendedaten*)
     DEC : ARRAY [0..20] OF UINT;  (* Baudrate, Timeout,..*)
     SEQ : ARRAY [0..16] OF BYTE;  (* Sequenzen *)
     FLG : ARRAY [0..16] OF BYTE;  (* Start- u. Stop-Flags *)
     COM_STAT: INT:=0;             (* Status der Schnittstelle *)
     ASC : WORD;                   (* Kanal z.B. 0,1,2*)
     ZEICHEN : BYTE;               (* intern *)
     TOUT : WORD;                  (* intern *)
     SADR : DWORD;                 (* intern *)
     TASK : WORD:=0;               (* PLC-Task *)
     nSendeIndex : INT:=0;         (* intern *)
END_STRUCT
END_TYPE
```

STRING-X Befehle: (siehe Abschnitt "Beschreibung der Funktionen")

INIT_STRX	Initialisierung der Schnittstelle
SRD_SRBOX	Lesen eines Strings aus der RBOX
SWR_STBOX	Schreiben eines Strings in die TBOX
SWR_TBCC	Definition des TBOX-Bereiches über den die BCC-Prüfung erfolgen soll
SWR_RBCC	Definition des RBOX-Bereiches über den die BCC-Prüfung erfolgen soll

2.4 Übersicht der Flags und Definitionen

Für jede Instanz wird ein Datenblock angelegt.
Dies geschieht im PLC-Programm mit der Definition einer Variablen vom Typ "STRX".

```
TYPE STRX
STRUCT
     RBOX : ARRAY[0..255] OF BYTE; (* Empfangsdaten *)
     TBOX : ARRAY[0..255] OF BYTE; (* Sendedaten*)
     DEC : ARRAY [0..20] OF UINT;  (* Baudrate, Timeout,..*)
     SEQ : ARRAY [0..16] OF BYTE;  (* Sequenzen *)
     FLG : ARRAY [0..16] OF BYTE;  (* Start- u. Stop-Flags *)
     COM_STAT: INT:=0;             (* Status der Schnittstelle *)
     ASC : WORD;                   (* Kanal z.B. 0,1,2*)
     ZEICHEN : BYTE;               (* intern *)
     TOUT : WORD;                  (* intern *)
     SADR : DWORD;                 (* intern *)
     TASK : WORD:=0;               (* PLC-Task *)
     nSendeIndex : INT:=0;         (* intern *)
END_STRUCT
END_TYPE
```

Beispiel einer Variablendeklaration:
In einem AWL-Programm sollen 2 Instanzen von STRING-X angelegt werden.

```
PROGRAM PLC_PRG
VAR
     Instanz1 :  STRX;
     Instanz2 :  STRX;
END_VAR
:
:
```

HINWEIS:
Durch die Autodeklaration-Funktion von GRIPS_cp PLC wird das Anlegen von Variablen erheblich erleichtert. Bei der Auswahl des Datentyps wird der Datentyp „STRX“ in dem Abschnitt „Definierte Datentypen“ angeboten.

ACHTUNG:
Alle nachfolgend beschriebenen Struktur-Daten können geändert werden und haben direkten Einfluß auf die STRINGX-Funktionalität.

2.4.1 Initialisierungsarten

Mit dem Datenfeld TASK kann der Funktionsblock STRINGX beliebig im EMD-Multitasking-System aktiviert werden. Die Standardeinstellung ist so definiert, dass der Funktionsblock synchron zum PLC-Programm (d.h. im Datenaustausch) ausgeführt wird.

Optional ist die Initialisierung des Funktionsblocks STRINGX in der EMD-Kommunikationstask möglich. Dies wird durch Beschreiben des Datenfelds TASK mit dem Wert 1 erreicht.

EMD-Task	Wert	Vorteile
PLC-Task	0 (Standard)	Funktionsblock wird synchron zum PLC-Zyklus im Datenaustausch ausgeführt. Erreicht hohe Übertragungsraten
Kommunikationstask	1	Verlängert den Datenaustausch nicht Kommunikation ist unabhängig von der PLC mit fest definierten Antwortzeiten

ACHTUNG:
Soll der Funktionsblock STRINGX in der Kommunikationstask initialisiert werden, muss das Datenfeld TASK vor dem Aufruf der Funktion INIT_STRX mit dem Wert 1 belegt Werden.

ACHTUNG:
Werden gleichzeitig mehrere serielle Schnittstellen verwendet, so muss sichergestellt sein, dass alle in derselben EMD-Task initialisiert werden.

Beispielprogramm in AWL:

```
     LD          INIT
     EQ          0
     JMPCN       NT1              (* naechste Taste *)

     LD          8                (* DEFAULT-INIT *)
     ST          S0.DEC[0]

     LD          1
     ST          S0.TASK          (* STINGX in KOM-Task initialisieren *)

     LD          S0
     ADR
     INIT_STRX   _STR0
     LD          1
     ST          S0.FLG[0]        (* Enable SEQ1 *)
     ST          S0.FLG[1]        (* Enable SEQ2 *)

     LD          1
     ST          INIT

NT1:
```

2.4.2 Übertragungsgeschwindigkeiten

Der Funktionsblock STRINGX ist für eine Kommunikation im sog. Handshake-Betrieb ausgelegt, d.h. die Daten werden vom Kommunikationspartner quittiert.

In diesem Betrieb sind alle aufgeführten Baudraten möglich.

HINWEIS:
Werden von einem externen Gerät (z.B. PC, Modem usw.) ständig Daten an das EMD gesendet ohne quitiert zu werden, ist die Baudrate auf max 9600 begrenzt.

2.5 Standard-Initialisierung

Mit dem Anlegen einer Instanz wird die Datenstruktur mit Standardwerten gefüllt. Wird die Instanz mit der STRING-X-Funktion „INIT_STRX“ initialisiert, wird mit dem Wert 8 im Datenfeld DEC[STRX_DEF] die Standardeinstellung verwendet.

DEC[STRX_MODE]	0xB40F	19200 Baud 7E1
DEC[STRX_TIME]	15	150ms Timeout
DEC[STRX_RMAX]	255	max. Länge RBOX
DEC[STRX_TMAX]	255	max. Länge TBOX
DEC[STRX_SEQ1A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ1E]	7	zwei definierte Endezeichen
DEC[STRX_SEQ2A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ2E]	7	zwei definierte Endezeichen
DEC[STRX_SEQ3A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ3E]	7	zwei definierte Endezeichen
DEC[STRX_SEQ4A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ4E]	7	zwei definierte Endezeichen
SEQ[STRX_SEQ_A11]	49	1-tes. Startzeichen 1
SEQ[STRX_SEQ_A12]	50	2-tes. Startzeichen 2
SEQ[STRX_SEQ_E11]	51	1-tes. Stopzeichen 3
SEQ[STRX_SEQ_E12]	52	2-tes. Stopzeichen 4
SEQ[STRX_SEQ_A21]	53	1-tes. Startzeichen 5
SEQ[STRX_SEQ_A22]	54	2-tes. Startzeichen 6
SEQ[STRX_SEQ_E21]	55	1-tes. Stopzeichen 7
SEQ[STRX_SEQ_E22]	56	2-tes. Stopzeichen 8
SEQ[STRX_SEQ_A31]	97	1-tes. Startzeichen a
SEQ[STRX_SEQ_A32]	98	2-tes. Startzeichen b
SEQ[STRX_SEQ_E31]	99	1-tes. Stopzeichen c
SEQ[STRX_SEQ_E32]	100	2-tes. Stopzeichen d
SEQ[STRX_SEQ_A41]	101	1-tes. Startzeichen e
SEQ[STRX_SEQ_A42]	102	2-tes. Startzeichen f
SEQ[STRX_SEQ_E41]	103	1-tes. Stopzeichen g
SEQ[STRX_SEQ_E42]	104	2-tes. Stopzeichen h

2.6 Auswahl der Schnittstellenparameter

Die Serielle Schnittstelle kann mit unterschiedlichen Parametern initialisiert werden. Dazu gehören u.a. die Baudrate, die Parität, die Anzahl der Stopbits und der Timeout.

Die jeweiligen Parameter sind in dem Datenfeld DEC[STRX_MODE] bitweise codiert. Der Aufbau dieses Bit-Feldes ist im folgenden beschrieben.

HINWEIS:
Die hier beschriebenen Einstellungen werden an die BIOS-Funktionn „BCom_Init“ übergeben. Falls Sie weitere Informationen dazu benötigen, finden Sie diese im C-Programmierhandbuch EMD 707-100 "BIOS-Beschreibung"

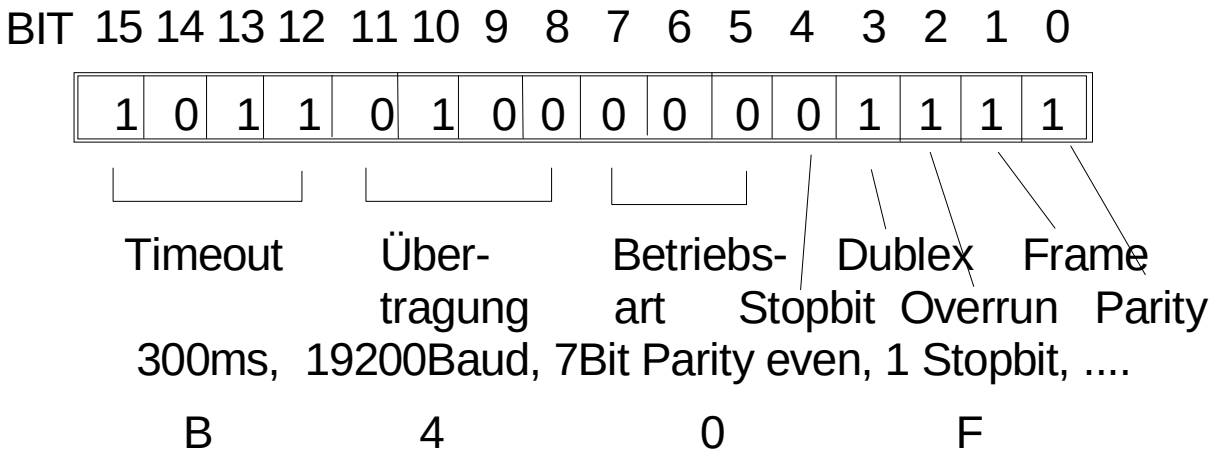
Schnittstellen-Mode (DEC[STRX_MODE])

Bit 0:	Freigabe Parity-Ueberwachung PARITY_ERROR_ENA	
Bit 1:	Freigabe Frame-Ueberwachung FRAME_ERROR_ENA	
Bit 2:	Freigabe Overrun-Ueberwachung OVERRUN_ERROR_ENA	
Bit 3:	Umschaltung Voll-Halbduplex	
	VOLLDUPLEX,	0
	HALBDUPLEX,	1
Bit 4:	Anzahl Stopbits	
	_1STOPBIT,	0
	_2STOPBIT,	1
Bit 5..7:	Betriebsart	
	_7BIT_PARITYEVEN,	000
	_7BIT_PARITYODD,	001
	_8BIT_NOPARITY,	010
	_8BIT_PARITYEVEN,	011
	_8BIT_PARITYODD,	100
Bit 8..11:	Uebertragungsgeschwindigkeit	
	_1200BAUD,	0000
	_2400BAUD,	0001
	_4800BAUD,	0010
	_9600BAUD,	0011
	_19200BAUD,	0100
	_38400BAUD,	0101
	_57600BAUD	0110

Auswahl der Schnittstellenparameter

Bit 12..15: Timeoutueberwachung Input

NO_TIMEOUT,	0000
TIMEOUT_10MS,	0001
TIMEOUT_20MS,	0010
TIMEOUT_30MS,	0011
TIMEOUT_40MS,	0100
TIMEOUT_50MS,	0101
TIMEOUT_75MS,	0110
TIMEOUT_100MS,	0111
TIMEOUT_150MS,	1000
TIMEOUT_200MS,	1001
TIMEOUT_250MS,	1010
TIMEOUT_300MS	1011
TIMEOUT_400MS,	1100
TIMEOUT_500MS,	1101
TIMEOUT_750MS,	1110
TIMEOUT_1000MS,	1111



2.7 Bearbeitung und Inhalt des Datenfeld DEC

Das Datenfeld DEC wird mit den Befehlen LD und ST angesprochen.

HINWEIS:
Bei dem Datenfeld DEC handelt es sich um ein ARRAY OF INT. Somit erfolgt der Zugriff auf die einzelnen Elemente über einen Feld-Index. In der Bibliothek „9010.lib“ sind zur einfachen Programmierung bereits Namen für die einzelnen Felder vorgegeben. Zur übersichtlichen Programmierung empfiehlt es sich, diese Namen zu verwenden. Die folgende Dokumentation der Bibliothek „9010.lib“ verwendet diese Namen.

Beispiel:

```
PROGRAM PLC_PRG
VAR
     S0 : STRX;        (* Variablendeklaration, Instanz *)
END_VAR
:
:



LD 8                   (* Default-Initialisierung *)
ST S0.DEC[STRX_DEF]
```

Bearbeitung und Inhalt des Datenfeld DEC

Inhalt von Datenfeld DEC

Das Datenfeld DEC besteht aus zwei Teilen:

- die Definitionen des STRING-x (Baudrate, Parity, Timeout,...)
  und
- Ergebnisrückmeldungen wie z.B. wieviele Zeichen empfangen wurden oder welchen Wert die BCC-Prüfung ermittelte.

Nachfolgend werden die einzelnen Positionen dieses Feldes beschrieben.

HINWEIS:
Die dem Feldindex vorangestellten Namen sind diejenigen, die in der Bibliothek definiert sind.

Bezeichnung	Index	Beschreibung
		Werte die vom Programmierer eingetragen werden
STRX_DEF	0	STRINGx Inhalt: 0 -> STRINGx ist nicht aktiv 1 -> STRINGx soll initialisiert werden (die Schnittstelle wird mit der in STRX_MODE vorhandenene Einstellung initialisiert). STRX_DEF enthält nach der Initialisierung den Wert 2 (auch wenn nicht initialisiert wurde steht Wert 2 drin und es kann aus FLGX_ERR Fehler ausgelesen werden) 2 -> STRINGx ist eingeschaltet 4 -> STRINGx soll ausgeschaltet werden, danach steht 0 in STRX_DEF 8 -> STRINGx soll mit default-Werten initialisiert werden. (die Schnittstelle wird mit der in STRX_MODE vorhandenen Einstellung initialisiert). STRX_DEF enthält nach der Initialisierung den Wert 2 (siehe Defaut-Initialisierung)
STRX_MODE	1	Baudrate,Datenbits, Stopbits, Parity,.. (siehe Abschnitt Schnittstellen-Mode)

Übertragungsgeschwindigkeiten

Bezeichnung	Index	Beschreibung
STRX_TIME	2	Timeout maximale Zeit zwischen zwei Zeichen. Einstellbar mit Vielfachen von 10ms (5-> 50ms, 20 -> 200ms) Die Taktzeit beträgt 10 ms. Erlaubte Werte sind deshalb 2 ... 32767

Wert	Zeit in ms
2	20
.	.
5	50
.	.
32767	327670

Werte die vom Programmierer eingetragen werden
(Werte für Index 3 und 4 müssen eingetragen werden, Index 5 bis 12 sind freigestellt)

STRX_RMAX	3	Länge RBox Größe der Empfangs-Box in Byte (max. 256)	
STRX_TMAX	4	Länge TBox Größe der Sende-Box in Byte (max. 256)	
STRX_SEQ1A	5	Sequenz 1 Start	(Werte siehe Kap. 2.8)
STRX_SEQ1E	6	Sequenz 1 Stop	(Werte siehe Kap. 2.8)
STRX_SEQ2A	7	Sequenz 2 Start	(Werte siehe Kap. 2.8)
STRX_SEQ2E	8	Sequenz 2 Stop	(Werte siehe Kap. 2.8)
STRX_SEQ3A	9	Sequenz 3 Start	(Werte siehe Kap. 2.8)
STRX_SEQ3E	10	Sequenz 3 Stop	(Werte siehe Kap. 2.8)
STRX_SEQ4A	11	Sequenz 4 Start	(Werte siehe Kap. 2.8)
STRX_SEQ4E	12	Sequenz 4 Stop	(Werte siehe Kap. 2.8)

Werte die vom System eingetragen werden

STRX_ANZR	13	Anzahl Zeichen in RBox Anzahl der empfangenen Zeichen
STRX_ANZT	14	Anzahl Zeichen in TBox Anzahl der zu sendenden Zeichen (wird vom Programmierer eingetragen)
STRX_SEQN	15	Nummer der empfangenen Sequenz (1,2,3,4)
STRX_BCC	16	Ergebnis BCC-Prüfung

2.8 Definition von Start- und Stop-Sequenzen

verfügbare Start-Sequenzen:

		dez.	bin.
SEQA_1BEL	ein beliebiges Startzeichen	8	1000
SEQA_1BEL_OHNESTOP	ein beliebiges Startzeichen, kein Stopzeichen, zum Übertragen von Einzelzeichen	9	1001
SEQA_1DEF	ein fest definiertes Startzeichen	10	1010
SEQA_1BEL_2BEL	zwei beliebige Startzeichen	12	1100
SEQA_1BEL_2DEF	zwei Startzeichen; erstes Startzeichen beliebig, zweites Startzeichen fest definiert	13	1101
SEQA_1DEF_2BEL	zwei Startzeichen; erstes Startzeichen fest definiert, zweites Startzeichen beliebig	14	1110
SEQA_1DEF_2DEF	zwei Startzeichen; erstes und zweites Startzeichen fest definiert	15	1111

verfügbare Stop-Sequenzen:

		dez.	din.
SEQE_1DEF	ein fest definiertes Stopzeichen	2	0010
SEQE_1DEF_2BEL	zwei Stopzeichen; erstes Stopzeichen fest definiert, zweites Stopzeichen beliebig)	6	0110
SEQE_1DEF_2DEF	zwei fest definierte Stopzeichen	7	0111
SEQE_KEINES	kein Stopzeichen Stop-Merker wird durch Timeout oder erreichen der max.Länge der RBOX gesetzt	8	1000

HINWEIS:
Die Werte für die Start- und Stop-Sequenzen werden im Datenfeld DEC in den Index 5 bis Index 12 eingetragen. (Strx.Dec[5] ... Strx.Dec[12])

2.9 Bearbeitung und Inhalt des Datenfeld SEQ

Das Datenfeld SEQ wird mit den Befehlen LD und ST angesprochen.

Bei dem Datenfeld SEQ handelt es sich um ein ARRAY OF BYTE. Somit erfolgt der Zugriff auf die einzelnen Elemente über einen Feld-Index.

Inhalt von SEQ:

In diesem Datenfeld werden die ASCII-Codes für die Start- u. Stopzeichen jeder Sequenz definiert.
Nachfolgend werden die einzelnen Positionen dieses Feldes beschrieben:

HINWEIS:
Die dem Feldindex vorangestellten Namen sind diejenigen, die in der Bibliothek definiert sind.

Bezeichnung	Index	Beschreibung
SEQX_A11	0	Sequenz-1 Startzeichen 1
SEQX_A12	1	Sequenz-1 Startzeichen 2
SEQX_E11	2	Sequenz-1 Stopzeichen 1
SEQX_E12	3	Sequenz-1 Stopzeichen 2
SEQX_A21	4	Sequenz-2 Startzeichen 1
SEQX_A22	5	Sequenz-2 Startzeichen 2
SEQX_E21	6	Sequenz-2 Stopzeichen 1
SEQX_E22	7	Sequenz-2 Stopzeichen 2
SEQX_A31	8	Sequenz-3 Startzeichen 1
SEQX_A32	9	Sequenz-3 Startzeichen 2
SEQX_E31	10	Sequenz-3 Stopzeichen 1
SEQX_E32	11	Sequenz-3 Stopzeichen 2
SEQX_A41	12	Sequenz-4 Startzeichen 1
SEQX_A42	13	Sequenz-4 Startzeichen 2
SEQX_E41	14	Sequenz-4 Stopzeichen 1
SEQX_E42	15	Sequenz-4 Stopzeichen 2

Beispiel:

```
PROGRAM PLC_PRG
VAR
     S0 : STRX;        (* Variablendeklaration, Instanz *)
END_VAR
:
:

     LD 49             (* ASCII 1 *)
     ST S0.SEQ[0]
     LD 50             (* ASCII 2 *)
     ST S0.SEQ[1]
     LD 51             (* ASCII 3 *)
     ST S0.SEQ[2]
     LD 52             (* ASCII 4 *)
     ST S0.SEQ[3]
```

2.10 Bearbeitung und Inhalt des Datenfeld FLG

Das Datenfeld FLG wird mit den Befehlen LD und ST angesprochen.

HINWEIS:
Bei dem Datenfeld FLG handelt es sich um ein ARRAY OF BYTE. Somit erfolgt der Zugriff auf die einzelnen Elemente über einen Feld-Index. In der Bibliothek „9010.lib“ sind zur einfachen Programmierung bereits Namen für die einzelnen Felder vorgegeben. Zur übersichtlichen Programmierung empfiehlt es sich, diese Namen zu verwenden. Die folgende Dokumentation der Bibliothek „9010.lib“ verwendet diese Namen.

Inhalt von FLG:

Das Datenfeld FLG besteht aus zwei Teilen:

- Definitionen die vom Programmierer festgelegt werden
  und
- Ergebnisrückmeldungen wie z.B. ob eine Stopsequenz erkannt wurde.

Nachfolgend werden die einzelnen Positionen dieses Feldes beschrieben:

1= Enable, 2= Disable

HINWEIS:
Die dem Feldindex vorangestellten Namen sind diejenigen, die in der Bibliothek definiert sind.

Bezeichnung	Index	Beschreibung
		Vom Programmierer einzutragen:
FLGX_ENS1	0	Enable Sequenz-1 Einschalten der Sequenz-1 Erkennung
FLGX_ENS2	1	Enable Sequenz-2 Einschalten der Sequenz-2 Erkennung
FLGX_ENS3	2	Enable Sequenz-3 Einschalten der Sequenz-3 Erkennung
FLGX_ENS4	3	Enable Sequenz-4 Einschalten der Sequenz-4 Erkennung
		Wird vom System eingetragen:
FLGX_STA	4	Start-Sequenz erkannt
FLGX_STO	5	Stop-Sequenz erkannt
FLGX_TIM	6	Timeout erkannt
FLGX_LNG	7	Länge erreicht erkannt
FLGX_STAT	8	Status erstes Start/Stop... erkannt
FLGX_MS1	9	Merker Sequenz-1

		vom System temporär verwaltet
FLGX_MS2	10	Merker Sequenz-2 **vom System temporär verwaltet**
FLGX_MS3	11	Merker Sequenz-3 **vom System temporär verwaltet**
FLGX_MS4	12	Merker Sequenz-4 **vom System temporär verwaltet**

Bezeichnung	**Index**	**Beschreibung**
FLGX_OUT	13	Starte Output über TBOX Inhalt: 0 Ausgabe beendet 1 Ausgabe der TBOX starten
FLGX_ERR	14	Fehler beim Empfang oder bei der Schnittstelleninitialisierung Inhalt: 0 kein Fehler 1 Parity-Error 2 Frame-Error 8 Falsche Kanal-Nummer oder ungültige Werte im Datenfeld DEC[STRX_MODE]
FLGX_BCC	15	BCC-Berechnung Inhalt: 1 Starte BCC Berechnung 0 BCC-Berechnung ist beendet

HINWEIS:
Ein erkannter Fehlerzustand bleibt erhalten, bis der Programmierer das Flag löscht.

ACHTUNG:
Die Datenfelder **FLGX_MS1 - FLGX_MS4** werden vom Funktionsblock STRING-X intern verwendet und dürfen durch das SPS-Programm nicht beschrieben werden !

Beispiel:

```
PROGRAM PLC_PRG
VAR
     S0 : STRX;        (* Variablendeklaration, Instanz *)
END_VAR
:
:

LD 1
ST S0.FLG[0]           (* enable Sequenz 1 *)
ST S0.FLG[1]           (* enable Sequenz 2 *)
```

2.11 Senden und Empfangen von Daten

Das Senden bzw. Empfangen von Daten mit dem Funktionsblock STRING-X läuft immer nach dem gleichen Schema ab:

Initialisieren der Schnittstelle

- Instanz vom Typ STRX anlegen
- Definitionen in die Variable eintragen
- Initialisierung der Schnittstelle
- Programm mit Lesen und Senden von Sequenzen

Senden von Daten

- Eine Sequenz mit Startzeichen, Daten, Endezeichen in die .TBOX[] schreiben. Je nach Kommunikationspartner sind die einzutragenden Daten von dem zu verwendenden Protokoll abhängig. Es können auch beliebige Zeichen ohne Start- u. Endezeichen eingetragen werden.

- Die Anzahl der Zeichen die gesendet werden sollen, müssen in dem Datenfeld DEC[STRX_ANZT] eingetragen werden.

- Zum Senden wird dann das SendenFlag FLG[FLGX_OUT] gesetzt. Dieses wird vom System zurückgesetzt nachdem der String gesendet wurde.

Empfangen von Daten

- Den Rahmen der Sequenz festlegen.
  Dabei werden die Anzahl und das Format der Start- u. Stopzeichen in den Datenfelder DEC[STRX_SEQ1A] und DEC[STRX_SEQ1E] festgelegt.

- Zu empfangende Sequenz mit Start und Stopzeichen definieren
  Dabei werden im Datenfeld SEQ[SEQX_A11], SEQ[SEQX_A12], SEQ[SEQX_E11] und SEQ[SEQX_E12] die Zeichen der Start- und Stopsequenz definiert.

- Die Sequenz im Datenfeld FLG[FLGX_ENS1] freischalten.

- Die Erkennung, ob Daten angekommen sind erfolgt über das Stop-Flag im Datenfeld FLG[FLGX_STO].
  Ist dieses gesetzt ist eine Sequenz angekommen.
  Welche Sequenz erkannt wurde kann im Datenfeld DEC[STRX_SEQN] gelesen werden.
  Die Anzahl der empfangenen Zeichen kann aus dem Datenfeld DEC[STRX_ANZR] gelesen werden.

- Um wieder neue Daten empfangen zu können müssen die Flags FLG[FLGX_STA], FLG[FLGX_STO], FLG[FLGX_TIM] und FLG[FLGX_LNG] gelöscht werden.

2.12 Fehlererkennung

Nach der Initialisierung sollte der Status im Datenfeld COM_STAT überprüft werden.

COM_STAT	0	bedeutet die Schnittstelle konnte korrekt initialisiert werden.
	1	bedeutet die Schnittstelle konnte nicht initialisiert werden.

Nach dem Empfangen von Daten kann in den Datenfelder FLG[FLGX_TIM], FLG[FLGX_LNG] und FLG[FLGX_ERR] der Zustand des Funktionsblocks STRING-X ermittelt werden.

FLG[FLGX_TIM]	1	Timeout erkannt
	0	Timeout nicht erkannt
FLG[FLGX_LNG]	1	Länge erreicht erkannt
	0	Länge erreicht wurde nicht erkannt
FLG[FLGX_ERR]	1	Fehler beim Empfang oder der Schnittstelleninit.
	0	kein Fehler beim Empfang oder der Schnittstelleninit.

2.13 Kommunikation mit StringX

2.13.1 Beschreibung der Funktionen

2.13.1.1 INIT_STRX

Funktion: Initialisierung einer Instanz

Mit dieser Funktion wird einer Instanz eine Schnittstelle Kanal 0,1, 2 zugeordnet und initialisiert. Die Daten der angelegten Instanz müssen zuvor definiert sein. Hier wird die Default-Initialisierung angeboten.

Daten die nicht die Schnittstellendefinition (Baud,Parity,..),
also den MODE betreffen, können geändert werden.

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Kanal	INT	0,1,2

ACHTUNG:
Die Funktion INIT_STRX verknüpft die String-Funktionalität mit dem Betriebssystem. Aus diesem Grund darf die Funktion für jede Instanz (S0:STRX oder Test:STRX) nur einmal ausgeführt werden.

HINWEIS:
Der Init-Befehl darf nur einmalig ausgeführt werden. Werden jedoch Änderungen im Datenfeld DEC vorgenommen, muss nochmals neu initialisiert werden.

Beispiel:
Für die serielle Schnittstelle 0 soll eine Instanz S0 angelegt und initialisiert werden.

Variablen-Deklaration:

```
S0 : STRX;
K0 : INT:=0;
DEFAULT :INT:=8;
```

AWL-Programm:

```
(* Adresse des Strings laden und INIT_STRX übergeben *)
LD    S0
ADR
INIT_STRX         K0

LD    DEFAULT
ST    S0.DEC[0]
```

2.13.1.2 SWR_STBOX

Funktion: Einen ASCII-String in die T-Box schreiben

Mit dieser Funktion kann die Sende-Box (T-BOX) aus einem oder mehreren Strings aufgebaut werden.

Bei Lng > 0 wird genau die vorgegebene Anzahl Zeichen in die TBOX geschrieben.

Bei Lng = 0 wird bis zum String-Ende-Zeichen 0 oder dem Erreichen der Maximallänge von 255 Zeichen in die TBOX geschrieben.

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	Position in T-Box
Lng	INT	0..Länge T-Box

Beispiel:
Aufbau des Sende-Strings "EPIS". Die Startzeichen sind A und B, die Endezeichen sind C und D.

Variablen-Deklaration:

```
S0 : STRX;
SENDE : STRING(8) := 'ABEPISCD';
```

AWL-Programm:

```
(* Adresse des Strings laden und SWR_STBOX übergeben *)
LD SENDE
ADR
SWR_STBOX   S0.SADR, 0, 8
```

2.13.1.3 SRD_SRBOX

Funktion: Einen ASCII-String aus der R-Box lesen

Mit dieser Funktion können aus der Empfangs-Box (R-BOX) Strings mit definierbarer Länge gelesen, d.h. in ein vorgegebenes Feld geschrieben werden.
Der String kann ab jeder gewünschten Position gelesen werden. Soll die komplette R-BOX gelesen werden wird Index auf 0 gesetzt. Sollen nur "Daten" (ohne Startzeichen) gelesen werden muß der Index auf 1 oder 2 gesetz werden.

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	Position in R-Box
Lng	INT	0..Länge der R-Box

Beispiel:
Lesen des ganzen empfangenen Datensatzes. Mit den 2 definierten Startzeichen und den 2 definierten Stopzeichen.

Variablen-Deklaration:

```
S0 : STRX;
DS_EMPF : ARRAY[0..32] OF STRING;
STOP_FLAG : BYTE := 5;
LNG : INT;
```

AWL-Programm:

```
LD          S0.FLG[STOP_FLAG]
EQ          0
JMPCN       WEITER      (* noch nichts empfangen *)
(* Auswertung der Daten *)
LD          S0.DEC[13]  (* Anzahl der empf. Zeichen lesen *)
ST LNG

LD DS_EMPF
ADR
SRD_SRBOX S0.SADR, 0, LNG


(* wieder aus empfangsbereit schalten *)
LD    0
ST    S0.FLG[STOP_FLAG]


WEITER:  ....
```

2.13.1.4 SWR_TBCC

Funktion: BCC-Prüfung über einen definierbaren Bereich der T-BOX (Sende-Box)

Mit dieser Funktion kann der Bereich der BCC-Prüfung der Sende-Box (T-BOX) festgelegt werden.
Mit dem Setzen des BCC-Flags wird die BCC-Prüfung gestartet. Ist die BCC-Prüfung beendet wird das BCC-Flag vom System gelöscht und das Ergebnis wird im DEC-Bereich eingetragen.
Die BCC-Prüfung bezieht sich immer auf die zuletzt ausgeführte Bereichsfestlegung (SWR_TBCC oder SWR_RBCC)

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	Position in T-Box
Lng	INT	0..Länge T-Box

Die Funktion trägt das Berechnungsergebnis im Datenfeld DEC[STRX_BCC] ein.

Beispiel:

Variablen-Deklaration:

```
S0 : STRX;
BCC_FLAG : INT := 15;
BCC_DEC :INT := 16;
BCC_RESULT : BYTE;
```

AWL-Programm:

```
BCC_START:
     (* Bereich der Prüfung festlegen *)
     LD          TRUE
     SWR_TBCC    S0.SADR, 0, 10

     (* BCC-Prüfung starten *)
     LD          1
     ST          S0.FLG[BCC_FLAG]
     JMP         NEXT

BBC_END:
     LD          S0.FLG[BCC_FLAG]
     EQ          0
     JMPCN       NEXT
     LD    S0.DEC[BCC_DEC]
     ST BCC_RESULT
     ...
```

2.13.1.5 SWR_RBCC

Funktion: BCC-Prüfung über einen definierbaren Bereich der R-BOX (Empfangs-Box)

Mit dieser Funktion kann der Bereich der BCC-Prüfung der Empfangs-Box (R-BOX) festgelegt werden.
Mit dem Setzen des BCC-Flags wird die BCC-Prüfung gestartet. Ist die BCC-Prüfung beendet wird das BCC-Flag vom System gelöscht und das Ergebnis wird im DEC-Bereich eingetragen.
Die BCC-Prüfung bezieht sich immer auf die zuletzt ausgeführte Bereichsfestlegung (SWR_TBCC oder SWR_RBCC)

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	Position in R-Box
Lng	INT	0..Länge R-Box

Die Funktion trägt das Berechnungsergebnis im Datenfeld DEC[STRX_BCC] ein.

Beispiel:

Variablen-Deklaration:

```
S0 : STRX;
BCC_FLAG : INT := 15;
BCC_DEC :INT := 16;
BCC_RESULT : BYTE;
```

AWL-Programm:

```
BCC_START:
     (* Bereich der Prüfung festlegen *)
     LD          TRUE
     SWR_RBCC    S0.SADR, 0, 10

     (* BCC-Prüfung starten *)
     LD          1
     ST          S0.FLG[BCC_FLAG]
     JMP         NEXT

BBC_END:
     LD          S0.FLG[BCC_FLAG]
     EQ          0
     JMPCN       NEXT
     LD          S0.DEC[BCC_DEC]
     ST          BCC_RESULT
     ...
```

Beispiel-Programm:
Mit der EMD-Taste F1 wird ein String gesendet und empfangene Daten werden immer angezeigt.

Variablen-Deklaration:

```
S0      : STRX;
_STR0 : INT:=0;
Text1 : STRING(10) := 'ein Text';
ENDE_ZEICHEN : INT := 0;
INIT : INT := 0;
```

AWL-Programm:

```
LD          TRUE
BUSY
JMPC        PRG_ENDE

LD          TRUE
KEY
ST          Taste

LD          TRUE
FNT         1


(* DEFAULT      I N I T I A L I S I E R U N G        *)
(*-------------------------------------------------*)
LD          INIT
EQ          0
JMPCN       NT1             (* naechste Taste *)

LD          S0
ADR
INIT_STRX   _STR0
LD          8               (* DEFAULT-INIT *)
ST          S0.DEC[0]
LD          1
ST          S0.FLG[0]       (* Enable SEQ1 *)
ST          S0.FLG[1]       (* Enable SEQ2 *)

LD          1
ST          INIT

NT1:
LD          Taste
EQ          31
JMPCN NT2
```

```
      (* S E N D E N  *)
      (* ------------ *)


      (* T-Box füllen und ausgeben *)

      LD          Text1
      ADR
      SWR_STBOX   S0.SADR,0,8       (* STRING-X Instance ; ; *)

      (* Anzahl der auszugebenden Zeichen setzen *)
      LD          8
      ST          S0.DEC[14]        (* STRING-X Instance ; ; *)

      (* Ausgabe-Flag setzen *)
      LD          1
      ST          S0.FLG[13]        (* Ausgabe-Flag ist gesetzt *)
      JMP         PRG_ENDE

NT2:

      (*  E M P F A N G E N  *)
      (* ------------------- *)
      LD          S0.FLG[5]         (* Stop-Sequenz erkannt ? *)
      EQ          1
      JMPCN PRG_ENDE
      LD          S0.DEC[13]        (* Anzahl der empfangenen Zeichen *)
      ST          ENDE_ZEICHEN

      LD          0                 (* Stringendezeichen *)
      ST          S0.RBOX[ENDE_ZEICHEN]

      LD          S0.RBOX[0]        (* Daten ins Display schreiben *)
      ADR
      TXT         17,8

      LD          0
      ST          S0.FLG[4]         (* StartFlag loeschen *)
      ST          S0.FLG[5]         (* StopFlag loeschen *)
      ST          S0.FLG[6]         (* TimeoutFlag loeschen *)
      ST          S0.FLG[7]         (* Länge erreicht Flag löschen *)

      LD          0
      ST          S0.DEC[13]        (* Anzahl der empf. Zeichen loeschen *)

PRG_ENDE:
```

Beispiel:
STRINGx auf Schnittstelle 0 mit Defaulteinstellungen definieren und die Sequenz 1 und 2 freigeben.

Variablen-Deklaration:

```
S0      : STRX;
```

AWL-Programm:

```
LD    S0
ADR
INIT_STRX
LD    8                 (* DEFAULT-INIT *)
ST    S0.DEC[0]
LD    1
ST    S0.FLG[0]         (* Enable SEQ1 *)
ST    S0.FLG[1]         (* Enable SEQ2 *)
```

Beispiel:
Lesen der Schnittstellendefinition (Mode)

Variablen-Deklaration:

```
S0 :  STRX;
MODE : INT:= 1;   (* Index des Parameterfeldes *)
COM :WORD;  (* Schnittstellendefinition *)
```

AWL-Programm:

```
LD    S0.DEC[MODE]
ST    COM
```

Beispiel:
Lesen der Nummer der empfangenen Sequenz

Variablen-Deklaration:

```
S0 : STRX;
SEQN : INT:= 15;  (* Index des Parameterfeldes *)
SEQ_NR :WORD; (* Sequenz-Nummer*)
```

AWL-Programm:

```
LD    S0.DEC[SEQN]
ST    SEQ_NR
```

Beispiel:
Definieren der Sequenz 1 des Stringx Kanal 0.

StringX		SEQUENZ-1
1.-tes Startzeichen	'A'	(65)
2.-tes Startzeichen	'B'	(66)
1.-tes Stopzeichen	'1'	(49)
2.-tes Stopzeichen	'2'	(50)

Variablen-Deklaration:

```
S0: STRX;
SEQ0_A11 : INT := 0;    (* Index Sequenz-1 Startzeichen 1 *)
SEQ0_A12 : INT := 1;    (* Index Sequenz-1 Startzeichen 2 *)
SEQ0_E11 : INT := 2;    (* Index Sequenz-1 Stoppzeichen 1 *)
SEQ0_E12 : INT := 3;    (* Index Sequenz-1 Stoppzeichen 2 *)
```

AWL-Programm:

```
LD 65
ST S0.SEQ[SEQ0_A11]
LD 66
ST S0.SEQ[SEQ0_A12]
LD 49
ST S0.SEQ[SEQ0_E11]
LD 50
ST S0.SEQ[SEQ0_E12]
```

Beispiel:
Lesen des 1.-ten Startzeichens und des 1.-ten Stopzeichens der Sequenz 1 des Stringx Kanal 0.

Variablen-Deklaration:

```
S0 STRX;
SEQ0_A11 : INT := 0;    (* Index Sequenz-1 Startzeichen 1 *)
SEQ0_E11 : INT := 2;    (* Index Sequenz-1 Stoppzeichen 1 *)
FIRST_START : BYTE;     (* erstes Startzeichen *)
FIRST_STOP  : BYTE;     (* erstes Stopzeichen *)
```

AWL-Programm:

```
LD    S0.SEQ[SEQ0_A11]
ST    FIRST_START
LD    S0.SEQ[SEQ0_E11]
ST    FIRST_STOP
```

Beispiel:
Einschalten der Sequenz 1 des String-0

Variablen-Deklaration:

```
S0: STRX;
FLG_ENS1 : INT := 0;    (* Index enable Sequenz-1  *)
```

AWL-Programm:

```
LD    1
ST    S0.FLG[FLG_ENS1]  (* Einschalten der Sequenz 1 *)
```

Beispiel:
Lesen des Stopzeichen-Flags und dann wieder auf Empfang schalten.

Variablen-Deklaration:

```
S0 : STRX;
CLR : BYTE :=0;
STOP_FLAG : BYTE := 5;
```

AWL-Programm:

```
LD    S0.FLG[STOP_FLAG]
EQ    1
JMPCN WARTEN
(* Auswertung der Daten *)
.....
LD    CLR
ST    S0.FLG[STOP_FLAG]

WARTEN:
D......
...
```

Beispiel:
Lesen des zweiten Zeichens der R-BOX, dies ist das erste Zeichen nach den Startzeichen (falls 2 Startzeichen definiert sind). Dieses Zeichen wird mit dem Zeichen 'E' (69) verglichen.

Variablen-Deklaration:

```
S0 : STRX;
```

AWL-Programm:

```
LD    S0.RBOX[2]
EQ    69
JMPCN ....
```

2.13.2 Beispielprogramm String

Globale Variablen:

```
VAR_GLOBAL
END_VAR
VAR_GLOBAL
     INIT : BYTE := 0;
     INITS1 :BYTE := 0;
     S0    : STRX;           (* STRING-X Instance *)
     S1    : STRX;
     S2    : STRX;

     STR0_HANDLE : WORD :=0;
     STR1_HANDLE : WORD :=1;
     STR2_HANDLE :WORD := 2;
END_VAR
```

Lokale Variablen:

```
PROGRAM PLC_PRG
VAR
     Taste: INT ;
     V_clr : BYTE:=0;
     ENDE_ZEICHEN : INT := 0;

     T1:   STRING(26) := 'DEFAULT INITIALISIERUNG';
     T2:   STRING(26) := 'AUSGABE IST GESTARTET';
     T3:   STRING(26) := 'Ablaufwechsel';
     T4:   STRING(13) := 'Text von EMD ';
     T5:   STRING(20);

     TX:   ARRAY[0..20] OF BYTE;
     TY:   ARRAY[0..20] OF BYTE;
     TZ:   ARRAY[0..20] OF BYTE;

     S_S0:INT:= 123;
     S_S1:INT:= 345;
     S_S2:INT:= 567;
     V1: INT;

END_VAR
```

AWL-Programm:

```
(* Nach dem Einschalten sind 3 Schnittstellen aktiv. Es sind dazu die *)
(* drei Instanzen S0,S1 u. S2 angelegt siehe globale Variable. *)
(* Bedienung:   *)
(* F2 : Sendet über S0 z.B. an ein Terminal "12()GHIJKL34"      *)
(* F3 : Falls Daten empfangen wurden, werden diese im Display angezeigt *)
(*   ebenso werden wichtige Flags der 3 Schnittst. FLG[] angezeigt, und   *)
(*   die Schnittstellen wieder zum Empfang freigeschaltet *)
(* F4 : Definiert die zwei Sequenzen der Schnittstelle 0 auf "d2 .. e4"
"f6..g8" um *)


(* Anmerkung:    Für Testzwecke wird in jeden empfangen Datensatz die Nummer
                 der Schnittstelle fix auf das 2.te Byte geschrieben,um zu
                 sehen welche Schnittstelle die Daten empfangen hat!! *)



     LD          TRUE
     BUSY
     JMPC  PRG_ENDE
     LD          TRUE
     MTF         30
     JMPCN z1
     LD          0
     ST          init
z1:
     LD          TRUE
     KEY
     ST          Taste

     LD          TRUE
     FNT         1



     (* DEFAULT      I N I T I A L I S I E R U N G        *)
     (*-------------------------------------------------------------*)


     LD          INIT
     EQ          0
     JMPCN       NT5              (* naechste Taste *)

     LD          8                (* DEFAULT-INIT *)
     ST          S0.DEC[0]        (* STRING-X Instance ; ; *)
     LD          S0               (* STRING-X Instance ; ; *)
     ADR
     INIT_STRX   STR0_HANDLE
     LD          1
     ST          S0.FLG[0]        (* STRING-X Instance ; ; Enable SEQ1 *)
     ST          S0.FLG[1]        (* STRING-X Instance ; ; Enable SEQ2 *)
     (*-----------------------------------*)
```

```
     LD          T1
     ADR
     TXT         5,8
     LD          1
     ST          INIT
     JMP         PRG_ENDE

NT5:
     LD          33
     EQ          Taste
     JMPCN NT0


     LD          1
     ST          INITS1
     LD          T4
     ADR
     TXT         3,8

NT0: (* STRINGx- Kanal-0 ausschalten mit Taste F1*)
     LD          29
     EQ          Taste
     JMPCN NT1
     LD          4
     ST          S0.DEC[0]         (* STRING-X Instance ; ; *)
     LD          t2
     ADR
     TXT_CLR     15,5

     (*Wenn Taste F2 gedrückt Daten an Terminal Senden*)
NT1:
     LD          30
     EQ          Taste
     JMPCN NT2


     (* S E N D E N  *)
     (* ----------------- *)

     (* T-Box füllen und ausgeben *)
     LD          t4
     ADR
     SWR_STBOX   s0.sadr,0,13            (* STRING-X Instance ; ; *)
     (* Anzahl der auszugebenden Zeichen setzen *)
     LD          13
     ST          S0.DEC[14]              (* STRING-X Instance ; ; *)

     (* Ausgabe-Flag setzen *)
     LD          1
     ST          S0.FLG[13]              (* STRING-X Instance ; ;Ausgabe-Flag
                                          ist gesetzt *)
     LD          T2
     ADR
     TXT         15,5
     JMP         PRG_ENDE

NT2:
```

```
PRG_ENDE:
     LD          S0.COM_STAT       (* STRING-X Instance ; ; *)
     ADR
     VAR_D '%4d',10,10
     LD          s0.dec[0]         (* STRING-X Instance ; ; *)
     ADR
     VAR_D '%4d',10,15


     (*****************************************************)
```

Beispielprogramm String

Kommunikation mit StringX

2.14 Serielle Kommunikation

Die BIOS-Funktionen der Funktionsgruppe „Serielle Kommunikation“ stellen dem Anwender Funktionen zur asynchronen Übertragung von Daten über eine serielle Schnittstelle (RS232 bzw. RS485) zur Verfügung. Durch die BIOS-Funktionen wird dabei kein festes Protokoll vorgegeben. Der Anwender kann ein beliebiges Protokoll mit diesen Funktionen aufbauen.

HINWEIS:
Bevor die Funktionen der Funktionsgruppe „Serielle Kommunikation“ benutzt werden können, muß mit der Funktion „BCom_init“ die serielle Schnittstelle initialisiert werden.

2.14.1 Beschreibung der BCom-Funktionen

2.14.1.1 Bcom_Init

Funktion: Initialisierung der SIO und FIFO's (Input und Output)

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2
Mode	WORD	Siehe Schnittstellen-parameter

HINWEIS:
Die Funktion BCom_init darf nur einmalig ausgeführt werden.

```
FUNCTION BCom_Init : INT
VAR_INPUT
     Enable:     BOOL; (* Funktion ausführen ? *)
     Kanal:      INT;  (* Kanalnummer, 1 - 2 *)
     Mode:       WORD; (* Schnittstellenparameter *)
END_VAR
```

Beschreibung: Initialisierung der SIO entsprechend der Übergabeparameter.

Die Schnittstelle ASC2 kann mit Handshake-Signalen RTS und CTS betrieben werden. Sollen diese Leitungen benutzt werden, dann muss die Schnittstelle im Vollduplex-Modus initialisiert werden.

Schnittstellenparameter:

Bit 0: Freigabe Parity-Ueberwachung
PARITY_ERROR_ENA

Bit 1: Freigabe Frame-Ueberwachung
FRAME_ERROR_ENA

Bit 2: Freigabe Overrun-Ueberwachung
OVERRUN_ERROR_ENA

Bit 3: Umschaltung Voll-Halbduplex
VOLLDUPLEX, 0
HALBDUPLEX, 1

Bit 4: Anzahl Stopbits
_1STOPBIT, 0
_2STOPBIT, 1

Bit 5..7: Betriebsart

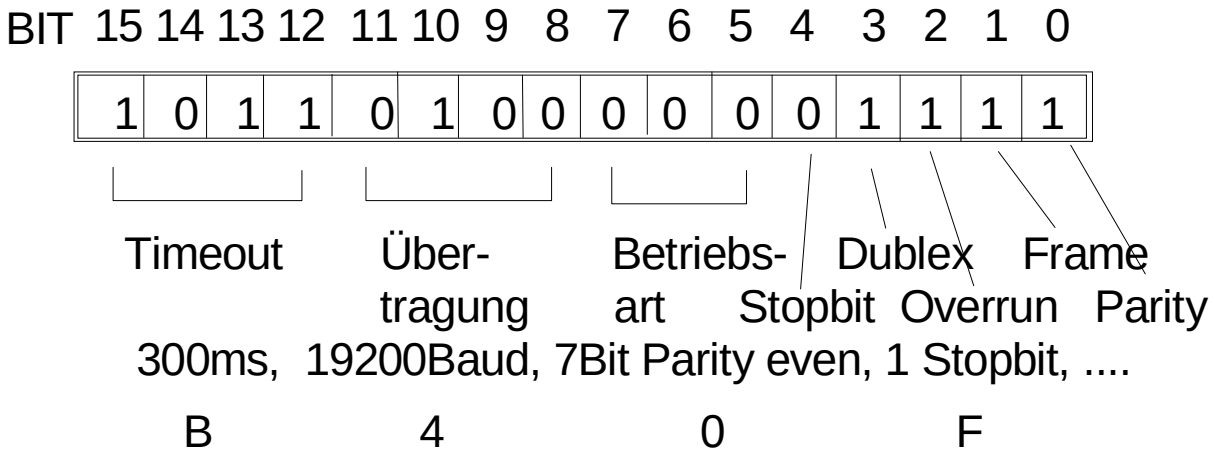
_7BIT_PARITYEVEN,	000
_7BIT_PARITYODD,	001
_8BIT_NOPARITY,	010
_8BIT_PARITYEVEN,	011
_8BIT_PARITYODD,	100

Bit 8..11: Uebertragungsgeschwindigkeit

_1200BAUD,	0000
_2400BAUD,	0001
_4800BAUD,	0010
_9600BAUD,	0011
_19200BAUD,	0100
_38400BAUD,	0101
_57600BAUD	0110

Bit 12..15: Timeoutueberwachung Input

NO_TIMEOUT,	0000
TIMEOUT_10MS,	0001
TIMEOUT_20MS,	0010
TIMEOUT_30MS,	0011
TIMEOUT_40MS,	0100
TIMEOUT_50MS,	0101
TIMEOUT_75MS,	0110
TIMEOUT_100MS,	0111
TIMEOUT_150MS,	1000
TIMEOUT_200MS,	1001
TIMEOUT_250MS,	1010
TIMEOUT_300MS	1011
TIMEOUT_400MS,	1100
TIMEOUT_500MS,	1101
TIMEOUT_750MS,	1110
TIMEOUT_1000MS,	1111



Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’
0	BIOS_OK:	wenn CTS-Signal aktiv (pos. Spannung an CTS)

Beispiel: Initialisierung der internen SIO mit folgenden Parametern: Baudrate: 19200 Baud, 7 Datenbits, Parity even, 1 Stopbit, Halbduplex, Überprüfung Parityerror, Überprüfung Frameerror, Überprüfung Overrunerror, Timeout Input vom 300 ms.

```
PROGRAM PLC_PRG
VAR
    Kanal:      INT:=1;     (*Kanalnummer der Schnittselle*)
    Mode:       WORD:=16#B40F;    (*Schnittstellenparameter*)
    init:       BOOL;       (*TRUE = Schnitstelle erfolgreich initalisiert*)
    InitReturn: INT;        (*Rückgabewert der Funkt. BCom_Init*)
END_VAR


    (** Schnittstelle Initalisieren **)
    LD          init
    JMPC        init_ok

    LD          TRUE
    BCom_Init   Kanal, Mode
    ST          InitReturn

    LD          InitReturn
    GE          0
    ST          init
init_ok:
```

2.14.1.2 COM_GetCts

Funktion: Zustand CTS-Leitung lesen

Parametertabelle:

Parameter	Typ	Werte
enable	BOOL	TRUE / FALSE
Kanal	UINT	1: ext. Schnittst. 1 2: ext. Schnittst. 2

Beschreibung: Von der über Kanal indizierten Schnittstelle wird der Zustand der CTS-Leitung geliefert:
Kanal = 1: entspr. Konstante ASC1 -> externe Schnittstelle 1
Kanal = 2: entspr. Konstante ASC2 -> externe Schnittstelle 2

```
FUNCTION COM_GetCts : INT
VAR_INPUT
     enable:          BOOL;
     Kanal:           UINT; (* Kanalnummer *)
END_VAR
```

Rückgabewerte: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1 BIOS_ERROR: Fehler ‘falscher Kanal’
1 wenn CTS-Signal aktiv (pos. Spannung an CTS)
0 wenn CTS-Signal inaktiv (neg. Spannung an CTS)

siehe auch: Com_SetRts

Beispiel: CTS-Signal von ASC2 als Handshake zum Senden verwenden:

```
PROGRAM PLC_PRG
VAR
     enable:           BOOL;
     Kanal:            UINT; (* Kanalnummer *)
     Zustand:          UINT;
END_VAR

(* ext. Schnittstelle 2 wählen *)
LD         2
ST         Kanal

(* RTS-Signal aktivieren *)
LD         1
ST         Zustand
LD         TRUE
COM_SetRts Kanal, Zustand

(* auf CTS-Signal warten)
LD         TRUE
COM_GetCts Kanal
EQ         0
JMPC       Weiter

     (* jetzt Sendevorgang starten *)

(* RTS-Signal deaktivieren *)
LD         0
ST         Zustand
LD         TRUE
COM_SetRts Kanal, Zustand
Weiter:
```

2.14.1.3 Com_setrts

Funktion: Zustand CTS-Leitung setzen

Parametertabelle:

Parameter	Typ	Werte
enable	BOOL	TRUE / FALSE
Kanal	UINT	1: ext. Schnittst. 1 2: ext. Schnittst. 2
Zustand	UINT	0: RTS inaktiv 1: RTS aktiv

Beschreibung: Von der über Kanal indizierten Schnittstelle wird der Zustand gemäß der Vorgabe in Zustand gesetzt:
Kanal = 1: entspr. Konstante ASC1 -> externe Schnittstelle 1
Kanal = 2: entspr. Konstante ASC2 -> externe Schnittstelle 2
Zustand = 0: setzt RTS inaktiv
Zustand = 1: setzt RTS aktiv

```
FUNCTION COM_SetRts : INT
VAR_INPUT
     enable:          BOOL;
     Kanal:           UINT; (* Kanalnummer *)
     Zustand:         INT;  (* 0=RTS inaktiv, 1=RTS aktiv *)
END_VAR
```

Rückgabewerte: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

0	BIOS_OK	
-1	BIOS_ERROR:	Fehler ‘falscher Kanal’
1	BIOS_ERROR:	Fehler ‘falscher Übergabewert’

siehe auch: Com_GetCts

Beispiel: CTS-Signal von ASC2 als Handshake zum Senden verwenden:

```
PROGRAM PLC_PRG
VAR
     enable:           BOOL;
     Kanal:            UINT; (* Kanalnummer *)
     Zustand:          UINT;
END_VAR

(* ext. Schnittstelle 2 wählen *)
LD         2
ST         Kanal

(* RTS-Signal aktivieren *)
LD         1
ST         Zustand
LD         TRUE
COM_SetRts Kanal, Zustand

(* auf CTS-Signal warten)
LD         TRUE
COM_GetCts Kanal
EQ         0
JMPC       Weiter

(* jetzt Sendevorgang starten *)

(* RTS-Signal deaktivieren *)
LD         0
ST         Zustand
LD         TRUE
COM_SetRts Kanal, Zustand
Weiter:
```

2.14.1.4 BCom_GetCTS

Funktion: Zustand CTS-Leitung lesen

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE / FALSE
Kanal	UINT	1: ext. Schnittst. 1 2: ext. Schnittst. 2

Beschreibung: Von der über Kanal indizierten Schnittstelle wird der Zustand der CTS-Leitung geliefert:
Kanal = 1: entspr. Konstante ASC1 -> externe Schnittstelle 1
Kanal = 2: entspr. Konstante ASC2 -> externe Schnittstelle 2

```
FUNCTION BCOM_GetCts : INT
VAR_INPUT
     enable:          BOOL;
     Kanal:           UINT; (* Kanalnummer *)
END_VAR
```

Rückgabewerte: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1 BIOS_ERROR: Fehler ‘falscher Kanal’
1 wenn CTS-Signal aktiv (pos. Spannung an CTS)
0 wenn CTS-Signal inaktiv (neg. Spannung an CTS)

siehe auch: BCom_SetRts

Beispiel: CTS-Signal von ASC2 als Handshake zum Senden verwenden:

```
PROGRAM PLC_PRG
VAR
     Enable:           BOOL;
     Kanal:            UINT; (* Kanalnummer *)
     Zustand:          UINT;
END_VAR

(* ext. Schnittstelle 2 wählen *)
LD         2
ST         Kanal

(* RTS-Signal aktivieren *)
LD         1
ST         Zustand
LD         TRUE
BCOM_SetRts      Kanal, Zustand

(* auf CTS-Signal warten)
LD         TRUE
BCOM_GetCts      Kanal
EQ         0
JMPC       Weiter

     (* jetzt Sendevorgang starten *)

(* RTS-Signal deaktivieren *)
LD         0
ST         Zustand
LD         TRUE
BCOM_SetRts      Kanal, Zustand
Weiter:
```

2.14.1.5 Bcom_SetRTS

Funktion: Zustand RTS-Leitung setzen

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE / FALSE
Kanal	UINT	1: ext. Schnittst. 1 2: ext. Schnittst. 2
Zustand	UINT	0: RTS inaktiv 1: RTS aktiv

Beschreibung: Von der über Kanal indizierten Schnittstelle wird der Zustand gemäß der Vorgabe in Zustand gesetzt:
Kanal = 1: entspr. Konstante ASC1 -> externe Schnittstelle 1
Kanal = 2: entspr. Konstante ASC2 -> externe Schnittstelle 2
Zustand = 0: setzt RTS inaktiv
Zustand = 1: setzt RTS aktiv

```
FUNCTION BCOM_SetRts : INT
VAR_INPUT
     Enable:          BOOL;
     Kanal:           UINT; (* Kanalnummer *)
     Zustand:         INT;  (* 0=RTS inaktiv, 1=RTS aktiv *)
END_VAR
```

Rückgabewerte: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

0	BIOS_OK	
-1	BIOS_ERROR:	Fehler ‘falscher Kanal’
1	BIOS_ERROR:	Fehler ‘falscher Übergabewert’

siehe auch: BCom_GetCts

Beispiel: RTS-Signal von ASC2 als Handshake zum Senden verwenden:

```
PROGRAM PLC_PRG
VAR
     Enable:           BOOL;
     Kanal:            UINT; (* Kanalnummer *)
     Zustand:          UINT;
END_VAR

(* ext. Schnittstelle 2 wählen *)
LD         2
ST         Kanal

(* RTS-Signal aktivieren *)
LD         1
ST         Zustand
LD         TRUE
BCOM_SetRts      Kanal, Zustand

(* auf CTS-Signal warten)
LD         TRUE
BCOM_GetCts      Kanal
EQ         0
JMPC       Weiter

(* jetzt Sendevorgang starten *)

(* RTS-Signal deaktivieren *)
LD         0
ST         Zustand
LD         TRUE
BCOM_SetRts      Kanal, Zustand
Weiter:
```

2.14.1.6 Bcom_Send

Funktion: Eintrag von einem oder mehreren Zeichen in die Output-FIFO.

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2
Quell_ptr	DWORD	Adresse der zu sendenden Zeichenkette
Anzahl	INT	Anzahl der zu sendenden Zeichen

Beschreibung: In die Output-FIFO werden entsprechend der Anzahl die übergebenen Zeichen eingetragen. Können nicht alle Zeichen in die Output-FIFO eingetragen werden, wartet die Funktion bis das letzte Zeichen eingetragen werden kann. Wenn die Ausgabe von der Funktion BCom_senddisable gesperrt ist, werden keine weiteren Zeichen in die Output-FIFO eingetragen.

```
FUNCTION BCom_Send : INT
VAR_INPUT
    Enable:     BOOL;       (* Funktion ausführen ? *)
    Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
    Quell_ptr:  DWORD;      (* Adresse der zu sendenden Zeichenkette*)
    Anzahl:     INT;        (* Anzahl der zu sendenden Zeichen*)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’
0	BIOS_OK:	OK

Beispiel: Programm trägt den in "Text" definierten String in den Outputfifo und versendet ihn anschließend.

```
Programm PLC_PRG
VAR
    Kanal:            INT   :=1;  (*Kanalnummer der Schnittselle*)
    AnzahlZeichenOut: INT:=5;     (*Anzahl der Sende Zeichen*)
    Text:             STRING(5):='Test';      (*Zu Sendender String*)
    QuelleAdr:        DWORD;      (*Quell Adresse von der gesendet
wird*)
END_VAR

    LD          Text
    ADR
    ST          QuelleAdr

    (* Eintrag und versenden der Zeichen im Output-Fifo *)
    LD          TRUE
    BCom_Send   Kanal, QuelleAdr, AnzahlZeichenOut
```

2.14.1.7 Bcom_SendDirect

Funktion: Direkte Ausgabe eines Zeichens mit Umgehung der Output-FIFO.

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2
Zeichen	INT	Zu sendendes Zeichen

Beschreibung: Direkte Ausgabe eines Zeichen mit Umgehung des Output-FIFO. Pro Funktionsaufruf kann maximal ein Zeichen an der Output-FIFO vorbei übertragen werden. Die Ausgabe erfolgt als nächstes Zeichen.

```
FUNCTION BCom_SendDirect : INT
VAR_INPUT
    Enable:     BOOL;       (* Funktion ausführen ? *)
    Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
    Zeichen:    INT;        (* zu sendendes Zeichen *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’, oder Ausgabespeicher belegt
0	BIOS_OK:	OK

Beispiel: Sendet ein Zeichen und umgeht dabei den Output-Fifo.

```
Programm PLC_PRG
VAR
    Kanal:            INT   :=1;  (*Kanalnummer der Schnittstelle*)
END_VAR

    (*Die Zeichenausgabe über den Outoput-FIFo wird unterbrochen*)
    LD          TRUE
    BCom_SendDisable  Kanal

    (*Direkte Ausgabe eines Zeichens mit Umgehung des Output-Fifo*)
    LD          TRUE
    BCom_SendDirect   Kanal, 16#41

    (*Freigabe der Ausgabe*)
    LD          TRUE
    BCom_SendEnable   KANAL
```

2.14.1.8 Bcom_SendDisable

Funktion: Die Zeichenausgabe über die Output-FIFO wird unterbunden.

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2

Beschreibung: Die Zeichenausgabe über die Output-FIFO wird unterbunden. Der Inhalt der Output-FIFO-Inhalt bleibt erhalten. Mit der Funktion „BCom_senddirect“ kann die Ausgabe trotz Unterbindung fortgesetzt werden.

```
FUNCTION BCom_SendDisable : INT
VAR_INPUT
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’, oder FIFO bereits disabled
0	BIOS_OK:	OK

Beispiel: siehe Bcom_SendDirect

2.14.1.9 Bcom_SendEnable

Funktion: Freigabe der Ausgabe (Output-FIFO)

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2

Beschreibung: Freigabe der Zeichenübertragung aus der Output-FIFO.

```
FUNCTION BCom_SendEnable : INT
VAR_INPUT
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’, oder FIFO bereits enabled
0	BIOS_OK:	OK

Beispiel: siehe Bcom_SendDirect

2.14.1.10 Bcom_Receive

Funktion: Auslesen von Zeichen aus der Input-FIFO

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2
Ziel_ptr	DWORD	Adresse der zu empfangenen Zeichenkette
MaxAnzahl	INT	Anzahl der zu empfangenen Zeichen
FifoFlag	INT	Fifopointer: 0= aktualisieren 1= unverändert lassen

Beschreibung: Auslesen der Input-FIFO.

```
FUNCTION BCom_Receive : INT
VAR_INPUT
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
     Ziel_ptr:   DWORD;      (* Adresse der zu sendenden Zeichenkette*)
     MaxAnzahl:  INT;        (* Anzahl der zu sendenden Zeichen*)
     FifoFlag:   INT;        (* Fifopointer aktualisieren(0) oder unverändert
lassen(1) *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’
0 .. Fifolänge:		Anzahl Zeichen die aus der Input-FIFO gelesen wurden

Beispiel: Liest die Zeichen aus dem Eingangsfifo aus und schreibt diese in das Array ZeichenIn.

```
Programm PLC_PRG
VAR
     Kanal:      INT   :=1;        (*Kanalnummer der Schnittselle*)
     AnzahlZeichenIn: INT:=10;     (*Anzahl der Empfangszeichen*)
     FiFoIn:     INT :=1;          (*Fifo Flag*)
     ZeichenIn:  ARRAY [1..10] OF BYTE;  (*Speicher der Empfangenen Zeichen*)
     ZielAdr:    DWORD;            (*Ziel Adresse an die, die Empf. Zeichen*)
                                   (*geschrieben werden*)
END_VAR

     LD          ZeichenIn[1]
     ADR
     ST          ZielAdr

     (* Auslesen der Zeichen aus dem Input-Fifo *)
     LD          TRUE
     BCom_Receive      Kanal, ZielAdr, AnzahlZeichenIn, FiFoIn
```

2.14.1.11 BCom_CheckFifo

Funktion: Überprüfung der FIFO's (Input oder Output)

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2
Fifo	INT	1= Input-Fifo 2= Output-Fifo

Beschreibung: Überprüfung der FIFO's. Bei dem Input-FIFO wird die Anzahl der eingetragenen Zeichen zurückgegeben. Bei dem Output-FIFO wird die Anzahl noch freier Eintragung zurückgegeben.

```
FUNCTION BCom_CheckFifo : INT
VAR_INPUT
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
     Fifo:       INT;        (* Input-Fifo(1) oder Output-Fifo(2) *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator
Bei Input: Anzahl der Zeichen der Input-Fifo
Bei Output: Anzahl der Zeichen die noch in die Output-Fifo eingetragen werden können.
-1 BIOS_ERROR: Fehler 'falscher Kanal'

Beispiel: Auslesen eines Zeichens aus der Input-Fifo

```
Programm PLC_PRG
VAR
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
     Fifo:       INT;        (* Input-Fifo(1) oder Output-Fifo(2) *)
End_Var

     (** Zyklisches Auslesen des Input-Fifos **)
     (* Anzahl Zeichen im Eingangs-Fifo *)
     LD          TRUE
     BCom_CheckFifo    Kanal, 1
     ST          AnzahlInFifo
```

2.14.1.12 BCom_ClearFifo

Funktion: Löschen der FIFO's (Input oder Output oder beide)

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2
Fifo	INT	1= Input-Fifo 2= Output-Fifo

Beschreibung: Rücksetzen der FIFO's

```
FUNCTION BCom_ClearFifo : INT
VAR_INPUT
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
     Fifo:       INT;        (* Input-Fifo(1) oder Output-Fifo(2) *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’, oder falsche FIFO-Anwahl
0	BIOS_OK:	OK

Beispiel: Eingangs-FIFO löschen.

```
Programm PLC_PRG
VAR
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
     Fifo:       INT;        (* Input-Fifo(1) oder Output-Fifo(2) *)
End_Var


     (* Eingangsfifo löschen*)
     LD          TRUE
     BCom_ClearFifo    Kanal, 1
```

2.14.1.13 Bcom_stat

Funktion: Überprüfung des SIO-Status und der FIFO's

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2

Beschreibung: Überprüft denSIO-Status und die FIFO's auf Übertragungsfehler.

```
FUNCTION BCom_Stat : INT
VAR_INPUT
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

Bit0: 1= Parity-Error
Bit1: 1= Frame-Error
Bit2: 1= Overrun-Error
Bit3: 1= Overflow Input-Fifo
Bit4: 1= Timeout abgelaufen
Bit5: 1= Input-Fifo empty
Bit6: 1= Output-Fifo empty
-1 BIOS_ERROR: Falscher Kanal

Beispiel: Programm prüft über die Funktion BCom_Stat den SIO-Status und die FIFO Flags. Tritt ein übertragungsfehler auf wird das Error-Bit gesetz und mit der Funktion BCom_ClearStat der alte SIO-Satus gelöscht. Im Folgezyklus wird das Error-Flag wieder zurück-genommen.

```
Programm PLC_PRG
VAR
     Kanal:            INT:=1;     (*Kanalnummer der Schnittstelle*)
     BComStatReturn:   WORD;       (*Rückgabewert der Funktion BCom_Stat*)
     BComErrors:       WORD;       (*Ergebnis nach der UND-Maskierung*)
     Error:            BOOL;       (*Error Flag*)
End_Var

     (*Überprüfung des SIO-Status und FIFO´s auf Übertragungsfehler*)
     LD          TRUE
     BCom_Stat   Kanal
     ST          BComStatReturn

     LD          BComStatReturn
     AND         2#0000000000000111
     ST          BComErrors

     LD          BComErrors
     GT          0
     JMPCN       No_Error

     LD          TRUE
     ST          Error
     JMP         Ende

No_Error:
     (*Löschen des Schnittstellen Status*)
     LD          TRUE
     BCom_ClearStat    Kanal

     LD          FALSE
     ST          Error
Ende:
```

2.14.1.14 Bcom_Clearstat

Funktion: Löschen des SIO-Status

Parametertabelle:

Parameter	Typ	Werte
Enable	BOOL	TRUE/FALSE
Kanal	INT	Kanalnummer 1-2

Beschreibung: Löschen des Schnittstellen-Status (SIO und FIFO-Fehler)

```
FUNCTION BCom_ClearStat : INT
VAR_INPUT
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
END_VAR
```

Rückgabewert: Nach Ausführung der Funktion steht der ret_code der C-Funktion im Akkumulator

-1	BIOS_ERROR:	Fehler ‘falscher Kanal’ oder falsche FIFO_Anwahll
0	BIOS_OK:	OK

Beispiel: Löschen des Schnittstellen-Status

```
Programm PLC_PRG
VAR
     Enable:     BOOL;       (* Funktion ausführen ? *)
     Kanal:      INT;        (* Kanalnummer, 1 - 2 *)
END_VAR


     LD          TRUE
     BCom_ClearStat    Kanal

     LD          FALSE
     ST          Error
```

2.14.2 Beispielprogramm

Beispiel zum Senden und Empfangen über die RS232-Schnittstelle unter Verwendung der Bcom-Funktionen.

```
PROGRAM PLC_PRG
VAR
     Kanal:      INT   :=1;        (*Kanalnummer der Schnittstelle*)
     Mode:       WORD:=16#B40F;    (*Schnittstellenparameter*)
     FiFoIn:     INT :=1;          (*Ffifo Flag*)
     AnzahlZeichenIn: INT:=10;     (*Anzahl der Empfangszeichen*)
     AnzahlZeichenOut: INT:=5;     (*Anzahl der Sende Zeichen*)
     init:       BOOL;             (*TRUE = Schnitstelle erfolgreich*)
                                   (*initalisiert*)
     InitReturn: INT;              (*Rückgabewert der Funkt. BCom_Init*)
     ZielAdr:    DWORD;            (*Ziel Adresse an die, die Empf. Zeichen*)
                                   (*geschrieben werden*)
     QuelleAdr:  DWORD;            (*Quell Adresse von der Gesendet wird*)
     Text  : STRING(5):='Test';    (*Zu Sendender String*)
     AnzahlInFifo: INT;            (*Anzahl der Empfangenen Zeichen*)
     ZeichenIn: ARRAY [1..10] OF BYTE;  (*Speicher der Empfangenen Zeichen*)
END_VAR



     (*****************************************************)
     (***** EMD PLC Hauptprogramm                    *****)
     (***** wird in jedem Zyklus aufgerufen         *****)
     (*****************************************************)

     (** Schnittstelle Initalisieren **)
     LD          init
     JMPC        init_ok

     LD          TRUE
     BCom_Init   Kanal, Mode
     ST          InitReturn

     LD          InitReturn
     GE          0
     ST          init
init_ok:

     (** Zyklisches Auslesen des Input-Fifos **)
     (* Anzahl Zeichen im Eingangsfifo *)
     LD          TRUE
     BCom_CheckFifo    Kanal, 1
     ST          AnzahlInFifo

     LD          AnzahlInFifo
     EQ          AnzahlZeichenIn
     JMPCN       NotClr

     (* Eingangsfifo löschen*)
     LD          TRUE
     BCom_ClearFifo    Kanal, 1
NotClr:

     LD          ZeichenIn[1]
     ADR
     ST          ZielAdr
```

```
      (* Auslesen der Zeichen aus dem Input-Fifo *)
      LD          TRUE
      BCom_Receive      Kanal, ZielAdr, AnzahlZeichenIn, FiFoIn

      LD          ZeichenIn[1]
      ADR
      VAR_D       '%d',1,1

      (** Einmaliges Senden des Output-Fifos mit der Info-Taste **)
      LD          TRUE
      MTF         14
      JMPCN       Ende

      LD          Text
      ADR
      ST          QuelleAdr

      (* Eintrag und versenden der Zeichen im Output-Fifo *)
      LD          TRUE
      BCom_Send   Kanal, QuelleAdr, AnzahlZeichenOut
Ende:
```

Beispielprogramm

Serielle Kommunikation

Inhaltsverzeichnis

	Index	Seite
3.1	Inhaltsverzeichnis	1
3.2	Allgemeines	3
3.3	STRING-X Datentyp und Befehle	4
3.4	Übersicht der Flags und Definitionen	5
3.5	Bearbeitung und Inhalt des Datenfeld DEC	6
3.6	Definition von Start- und Stop-Sequenzen	9
3.7	Bearbeitung und Inhalt des Datenfeld SEQ	10
3.8	Bearbeitung und Inhalt des Datenfeld FLG	11
3.9	Standard-Initialisierung	14
3.10	Auswahl der Schnittstellenparameter	15
3.11	Senden und Empfangen von Daten	17
3.12	Fehlererkennung	18
3.13	Kommunikation mit StringX	19
3.13.1	Beschreibung der Funktionen	19
3.13.2	Beispielprogramm String	24
3.14	Serielle Kommunikation	25
3.14.1	Allgemeines	25
3.14.2	Bcom Funktionsbeschreibung	26

Inhaltsverzeichnis

3.2 Allgemeines

Zur Programmierung von Kommunikations-Anwendungen über die serielle Schnittstelle wird dem Programmierer eine weitere Bibliothek zur Verfügung gestellt.

Die in diesem Abschnitt beschriebene Bibliothek „STRINGX.LIB“ stellt dem Anwender Funktionen zur Verfügung mit denen ein Protokoll aufgebaut werden kann, die Schnittstelle initialisiert wird und Daten gesendet bzw. empfangen werden können.

HINWEIS:
Während der STRING-0 aktiv ist, kann keine ONLINE-Verbindung mit GRIPS_PLC hergestellt werden. Es empfiehlt sich daher im PLC-Programm eine Möglichkeit zur Deaktivierung von STRING-0 vorzusehen.

Die Programmierung erfolgt im allgemeinen über die Datenschnittstelle, die durch die Datenstruktur „STRX“ vorgegeben ist. Lediglich für spezielle Funktionen, wie z.B. die BCC-Prüfung ist die Verwendung von vorgegebenen Funktionen notwendig.

3.3 STRING-X Datentyp und Befehle

STRING-X Datentyp

```
TYPE STRX
STRUCT
     RBOX : ARRAY[0..255] OF BYTE; (* Empfangsdaten *)
     TBOX : ARRAY[0..255] OF BYTE; (* Sendedaten*)
     DEC : ARRAY [0..20] OF INT;   (* Baudrate, Timeout,..*)
     SEQ : ARRAY [0..16] OF BYTE;  (* Sequenzen *)
     FLG : ARRAY [0..16] OF BYTE;  (* Start- u. Stop-Flags *)
     COM_STAT: INT:=0;             (* Status der Schnittstelle *)
     ASC : WORD;                   (* Kanal z.B. 0,1,2*)
     ZEICHEN : BYTE;                     (* intern *)
     TOUT : WORD;                        (* intern *)
     SADR : DWORD;                       (* intern *)
END_STRUCT
END_TYPE
```

STRING-X Befehle

INIT_STRX	Initialisierung der Schnittstelle
SRD_SRBOX	Lesen eines Strings aus der RBOX
SWR_STBOX	Schreiben eines Strings in die TBOX
SWR_TBCC	Definition des TBOX-Bereiches über den die BCC-Prüfung erfolgen soll
SWR_RBCC	Definition des RBOX-Bereiches über den die BCC-Prüfung erfolgen soll

3.4 Übersicht der Flags und Definitionen

Für jede Instanz wird ein Datenblock angelegt.
Dies geschieht im PLC-Programm mit der Definition einer Variablen vom Typ "STRX".

HINWEIS:
Durch die Autodeklaration-Funktion von GRIPS_cp PLC wird das Anlegen von Variablen erheblich erleichtert. Beim Auswahl des Datentyps wird der Datentyp „STRX“ in dem Abschnitt „Definierte Datentypen“ angeboten.

ACHTUNG:
Alle nachfolgend beschriebenen Struktur-Daten können geändert werden und haben direkten Einfluß auf die STRINGX-Funktionalität. Die Variable **MODE** im **ARRAY DEC** (mit dem Feldindex 1) kann nur einmal mit der Initalisierung INIT_STRX definiert werden.

```
TYPE STRX
STRUCT
     RBOX : ARRAY[0..255] OF BYTE; (* Empfangsdaten *)
     TBOX : ARRAY[0..255] OF BYTE; (* Sendedaten*)
     DEC : ARRAY [0..20] OF INT;   (* Baudrate, Timeout,..*)
     SEQ : ARRAY [0..16] OF BYTE;  (* Sequenzen *)
     FLG : ARRAY [0..16] OF BYTE;  (* Start- u. Stop-Flags *)
     COM_STAT: INT:=0;             (* Status der Schnittstelle *)
     ASC : WORD;                   (* Kanal z.B. 0,1,2*)
     ZEICHEN : BYTE;                    (* intern *)
     TOUT : WORD;                       (* intern *)
     SADR : DWORD;                      (* intern *)
END_STRUCT
END_TYPE
```

3.5 Bearbeitung und Inhalt des Datenfeld DEC

Das Datenfeld DEC wird mit den Befehlen LD und ST angesprochen.

HINWEIS:
Bei dem Datenfeld DEC handelt es sich um ein ARRAY OF INT. Somit erfolgt der Zugriff auf die einzelnen Elemente über einen Feld-Index. In der Bibliothek „STRINGX.LIB“ sind zur einfachen Programmierung bereits Namen für die einzelnen Felder vorgegeben. Zur übersichtlichen Programmierung empfiehlt es sich, diese Namen zu verwenden. Die folgende Dokumentation der Bibliothek „STRING-X“ verwendet diese Namen.

Bearbeitung und Inhalt des Datenfeld DEC

Inhalt von DEC

Das Datenfeld DEC besteht aus zwei Teilen:

- die Definitionen des STRING-x (Baudrate, Parity, Timeout,...)
  und
- Ergebnisrückmeldungen wie z.B. wieviele Zeichen empfangen wurden oder welchen Wert die BCC-Prüfung ermittelte.

Nachfolgend werden die einzelnen Positionen dieses Feldes beschrieben.

HINWEIS:
Die dem Feldindex vorangestellten Namen sind diejenigen, die in der Bibliothek definiert sind.

Bezeichnung	Index	Beschreibung
		Werte die vom Programmierer eingetragen werden
STRX_DEF	0	STRINGx Inhalt: 0 -> STRINGx ist nicht aktiv 1 -> STRINGx soll initialisiert werden (die Schnittstelle wird mit der in STRX_MODE vorhandenene Einstellung initialisiert). STRX_DEF enthält nach der Initialisierung den Wert 2 2 -> STRINGx ist eingeschaltet 4 -> STRINGx soll ausgeschaltet werden, danach steht 0 in STRX_DEF 8 -> STRINGx soll mit default-Werten initialisiert werden. (die Schnittstelle wird mit der in STRX_MODE vorhandenene Einstellung initialisiert). STRX_DEF enthält nach der Initialisierung den Wert 2 (siehe Defaut-Initialisierung)
STRX_MODE	1	Baudrate,Datenbits, Stopbits, Parity,.. (siehe Abschnitt Schnittstellen-Mode)
STRX_TIME	2	Timeout maximale Zeit zwischen zwei Zeichen. Einstellbar mit Vielfachen von 10ms (5 -> 50ms, 20 -> 200ms)

Bezeichnung	Index	Beschreibung
		Werte die vom Programmierer eingetragen werden
STRX_RMAX	3	Länge RBox Größe der Empfangs-Box in Byte (max. 256)
STRX_TMAX	4	Länge TBox Größe der Sende-Box in Byte (max. 256)
STRX_SEQ1A	5	Sequenz 1 Start
STRX_SEQ1E	6	Sequenz 1 Stop
STRX_SEQ2A	7	Sequenz 2 Start
STRX_SEQ2E	8	Sequenz 2 Stop
STRX_SEQ3A	9	Sequenz 3 Start
STRX_SEQ3E	10	Sequenz 3 Stop
STRX_SEQ4A	11	Sequenz 4 Start
STRX_SEQ4E	12	Sequenz 4 Stop
		Werte die vom System eingetragen werden
STRX_ANZR	13	Anzahl Zeichen in RBox Anzahl der empfangenen Zeichen
STRX_ANZT	14	Anzahl Zeichen in TBox Anzahl der zu sendenden Zeichen (wird vom Programmierer eingetragen)
STRX_SEQN	15	Nummer der empfangenen Sequenz (1,2,3,4)
STRX_BCC	16	Ergebnis BCC-Prüfung

3.6 Definition von Start- und Stop-Sequenzen

verfügbare Start-Sequenzen:

		dez.	bin.
SEQA_1BEL	ein beliebiges Startzeichen	8	1000
SEQA_1BEL_OHNESTOP	ein beliebiges Startzeichen, kein Stopzeichen, zum Übertragen von Einzelzeichen	9	1001
SEQA_1DEF	ein fest definiertes Startzeichen	10	1010
SEQA_1BEL_2BEL	zwei beliebige Startzeichen	12	1100
SEQA_1BEL_2DEF	zwei Startzeichen; erstes Startzeichen beliebig, zweites Startzeichen fest definiert	13	1101
SEQA_1DEF_2BEL	zwei Startzeichen; erstes Startzeichen fest definiert, zweites Startzeichen beliebig	14	1110
SEQA_1DEF_2DEF	zwei Startzeichen; erstes und zweites Startzeichen fest definiert	15	1111

verfügbare Stop-Sequenzen:

		dez	bin
SEQE_1DEF	ein fest definiertes Stopzeichen	2	0010
SEQE_1DEF_2BEL	zwei Stopzeichen; erstes Stopzeichen fest definiert, zweites Stopzeichen beliebig)	6	0110
SEQE_1DEF_2DEF	zwei fest definierte Stopzeichen	7	0111
SEQE_KEINES	kein Stopzeichen Stop-Merker wird durch Timeout oder erreichen der max.Länge der RBOX gesetzt	8	1000

3.7 Bearbeitung und Inhalt des Datenfeld SEQ

Das Datenfeld SEQ wird mit den Befehlen LD und ST angesprochen.

HINWEIS:
Bei dem Datenfeld SEQ handelt es sich um ein ARRAY OF BYTE. Somit erfolgt der Zugriff auf die einzelnen Elemente über einen Feld-Index. In der Bibliothek „STRING.LIB“ sind zur einfachen Programmierung bereits Namen für die einzelnen Felder vorgegeben. Zur übersichtlichen Programmierung empfiehlt es sich, diese Namen zu verwenden. Die folgende Dokumentation der Bibliothek „STRING-X“ verwendet diese Namen.

Inhalt von SEQ:

In diesem Datenfeld werden die ASCII-Codes für die Start- u. Stopzeichen jeder Sequenz definiert.

Nachfolgend werden die einzelnen Positionen dieses Feldes beschrieben:

HINWEIS:
Die dem Feldindex vorangestellten Namen sind diejenigen, die in der Bibliothek definiert sind.

Bezeichnung	Index	Beschreibung
SEQX_A11	0	Sequenz-1 Startzeichen 1
SEQX_A12	1	Sequenz-1 Startzeichen 2
SEQX_E11	2	Sequenz-1 Stopzeichen 1
SEQX_E12	3	Sequenz-1 Stopzeichen 2
SEQX_A21	4	Sequenz-2 Startzeichen 1
SEQX_A22	5	Sequenz-2 Startzeichen 2
SEQX_E21	6	Sequenz-2 Stopzeichen 1
SEQX_E22	7	Sequenz-2 Stopzeichen 2
SEQX_A31	8	Sequenz-3 Startzeichen 1
SEQX_A32	9	Sequenz-3 Startzeichen 2
SEQX_E31	10	Sequenz-3 Stopzeichen 1
SEQX_E32	11	Sequenz-3 Stopzeichen 2
SEQX_A41	12	Sequenz-4 Startzeichen 1
SEQX_A42	13	Sequenz-4 Startzeichen 2
SEQX_E41	14	Sequenz-4 Stopzeichen 1
SEQX_E42	15	Sequenz-4 Stopzeichen 2

3.8 Bearbeitung und Inhalt des Datenfeld FLG

Das Datenfeld FLG wird mit den Befehlen LD und ST angesprochen.

HINWEIS:
Bei dem Datenfeld FLG handelt es sich um ein ARRAY OF BYTE. Somit erfolgt der Zugriff auf die einzelnen Elemente über einen Feld-Index. In der Bibliothek „STRINGX.LIB“ sind zur einfachen Programmierung bereits Namen für die einzelnen Felder vorgegeben. Zur übersichtlichen Programmierung empfiehlt es sich, diese Namen zu verwenden. Die folgende Dokumentation der Bibliothek „STRING-X“ verwendet diese Namen.

Inhalt von FLG:

Das Datenfeld FLG besteht aus zwei Teilen:

- Definitionen die vom Programmierer festgelegt werden
  und
- Ergebnisrückmeldungen wie z.B. ob eine Stopsequenz erkannt wurde.

Nachfolgend werden die einzelnen Positionen dieses Feldes beschrieben:

HINWEIS:
Die dem Feldindex vorangestellten Namen sind diejenigen, die in der Bibliothek definiert sind.

Bezeichnung	Index	Beschreibung
FLGX_ENS1	0	Enable Sequenz-1 Einschalten der Sequenz-1 Erkennung
FLGX_ENS2	1	Enable Sequenz-2 Einschalten der Sequenz-2 Erkennung
FLGX_ENS3	2	Enable Sequenz-3 Einschalten der Sequenz-3 Erkennung
FLGX_ENS4	3	Enable Sequenz-4 Einschalten der Sequenz-4 Erkennung
FLGX_STA	4	Start-Sequenz erkannt
FLGX_STO	5	Stop-Sequenz erkannt
FLGX_TIM	6	Timeout erkannt
FLGX_LNG	7	Länge erreicht erkannt
FLGX_STAT	8	Status erstes Start/Stop... erkannt
FLGX_MS1	9	Merker Sequenz-1 **vom System temporär verwaltet**
FLGX_MS2	10	Merker Sequenz-2 **vom System temporär verwaltet**
FLGX_MS3	11	Merker Sequenz-3 **vom System temporär verwaltet**
FLGX_MS4	12	Merker Sequenz-4 **vom System temporär verwaltet**

Bearbeitung und Inhalt des Datenfeld FLG

Bezeichnung	Index	Beschreibung
FLGX_OUT	13	Starte Output über TBOX Inhalt: 0 Ausgabe beendet 1 Ausgabe der TBOX starten
FLGX_ERR	14	Fehler beim Empfang oder bei der Schnittstelleninitialisierung Inhalt: 0 kein Fehler 1 Parity-Error 2 Frame-Error 8 Falscher Kanal oder ungültige Werte im Datenfeld DEC[STRX_MODE]
FLGX_BCC	15	BCC-Berechnung Inhalt: 1 Starte BCC Berechnung 0 BCC-Berechnung ist beendet

HINWEIS:
Ein erkannter Fehlerzustand bleibt erhalten, bis der Programmierer das Flag löscht.

ACHTUNG:
Die Datenfelder **FLGX_MS1 - FLGX_MS4** werden vom Funktionsblock STRING-X intern verwendet und dürfen durch das SPS-Programm nicht beschrieben werden !

3.9 Standard-Initialisierung

Mit dem Anlegen einer Instanz wird die Datenstruktur mit Standardwerten gefüllt. Wird die Instanz mit der STRING-X-Funktion „INIT_STRX“ initialisiert, wird mit dem Wert 8 im Datenfeld DEC[STRX_DEF] die Standardeinstellung verwendet.

DEC[STRX_MODE]	0xB40F	19200 Baud 7E1
DEC[STRX_TIME]	15	150ms Timeout
DEC[STRX_RMAX]	255	max. Länge RBOX
DEC[STRX_TMAX]	255	max. Länge TBOX
DEC[STRX_SEQ1A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ1E]	7	zwei definierte Endezeichen
DEC[STRX_SEQ2A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ2E]	7	zwei definierte Endezeichen
DEC[STRX_SEQ3A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ3E]	7	zwei definierte Endezeichen
DEC[STRX_SEQ4A]	15	zwei definierte Startzeichen
DEC[STRX_SEQ4E]	7	zwei definierte Endezeichen
DEC[STRX_SEQ_A11]	49	1-tes. Startzeichen 1
DEC[STRX_SEQ_A12]	50	2-tes. Startzeichen 2
DEC[STRX_SEQ_E11]	51	1-tes. Stopzeichen 3
DEC[STRX_SEQ_E12]	52	2-tes. Stopzeichen 4
DEC[STRX_SEQ_A21]	53	1-tes. Startzeichen 5
DEC[STRX_SEQ_A22]	54	2-tes. Startzeichen 6
DEC[STRX_SEQ_E21]	55	1-tes. Stopzeichen 7
DEC[STRX_SEQ_E22]	56	2-tes. Stopzeichen 8
DEC[STRX_SEQ_A31]	97	1-tes. Startzeichen a
DEC[STRX_SEQ_A32]	98	2-tes. Startzeichen b
DEC[STRX_SEQ_E31]	99	1-tes. Stopzeichen c
DEC[STRX_SEQ_E32]	100	2-tes. Stopzeichen d
DEC[STRX_SEQ_A41]	101	1-tes. Startzeichen e
DEC[STRX_SEQ_A42]	102	2-tes. Startzeichen f
DEC[STRX_SEQ_E41]	103	1-tes. Stopzeichen g
DEC[STRX_SEQ_E42]	104	2-tes. Stopzeichen h

3.10 Auswahl der Schnittstellenparameter

Die Serielle Schnittstelle kann mit unterschiedlichen Parametern initialisiert werden. Dazu gehören u.a. die Baudrate, die Parität, die Anzahl der Stopbits und der Timeout.

Die jeweiligen Parameter sind in dem Datenfeld DEC[STRX_MODE] bitweise codiert. Der Aufbau dieses Bit-Feldes ist im folgenden beschrieben.

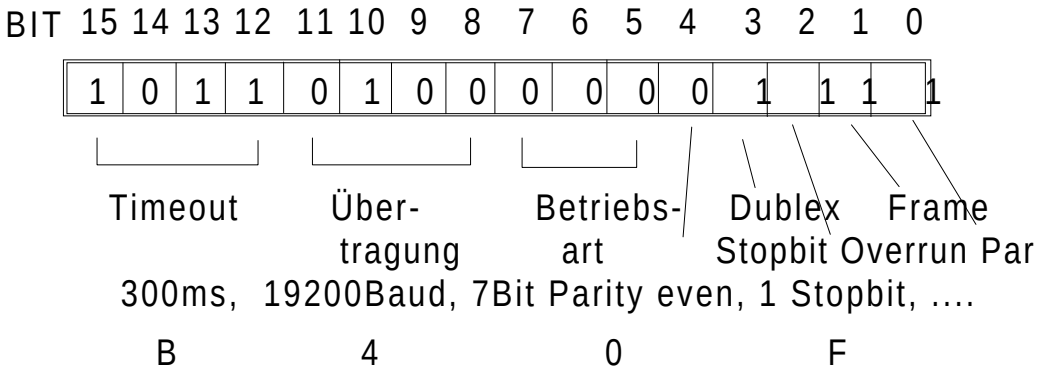
HINWEIS:
Die hier beschriebenen Einstellungen werden an die BIOS-Funktion „BCom_Init“ übergeben. Falls Sie weitere Informationen dazu benötigen, finden Sie diese im C-Programmierhandbuch EMD 707-100 "BIOS-Beschreibung"

Schnittstellen-Mode (DEC[STRX_MODE])

Bit 0:	Freigabe Parity-Ueberwachung	
	PARITY_ERROR_ENA	
Bit 1:	Freigabe Frame-Ueberwachung	
	FRAME_ERROR_ENA	
Bit 2:	Freigabe Overrun-Ueberwachung	
	OVERRUN_ERROR_ENA	
Bit 3:	Umschaltung Voll-Halbduplex	
	VOLLDUPLEX,	1
	HALBDUPLEX	0
Bit 4:	Anzahl Stopbits	
	_1STOPBIT,	0
	_2STOPBIT	1
Bit 5..7:	Betriebsart	
	_7BIT_PARITYEVEN,	000
	_7BIT_PARITYODD,	001
	_8BIT_NOPARITY,	010
	_8BIT_PARITYEVEN,	011
	_8BIT_PARITYODD	100
Bit 8..11:	Uebertragungsgeschwindigkeit	
	_1200BAUD,	0000
	_2400BAUD,	0001
	_4800BAUD,	0010
	_9600BAUD,	0011
	_19200BAUD,	0100
	_38400BAUD,	0101
	_57600BAUD	0110

Bit 12..15: Timeoutueberwachung Input

NO_TIMEOUT,	0000
TIMEOUT_10MS,	0001
TIMEOUT_20MS,	0010
TIMEOUT_30MS,	0011
TIMEOUT_40MS,	0100
TIMEOUT_50MS,	0101
TIMEOUT_75MS,	0110
TIMEOUT_100MS,	0111
TIMEOUT_150MS,	1000
TIMEOUT_200MS,	1001
TIMEOUT_250MS,	1010
TIMEOUT_300MS	1011
TIMEOUT_400MS,	1100
TIMEOUT_500MS,	1101
TIMEOUT_750MS,	1110
TIMEOUT_1000MS,	1111



3.11 Senden und Empfangen von Daten

Das Senden bzw. Empfangen von Daten mit dem Funktionsblock STRING-X läuft immer nach dem gleichen Schemas ab:

Initialisieren der Schnittstelle

- Instanz vom Typ STRX anlegen
- Definitionen in die Variable eintragen
- Initialisierung der Schnittstelle
- Programm mit Lesen und Senden von Sequenzen

Senden von Daten

- eine Sequenz mit Startzeichen, Daten, Endezeichen in die .TBOX[] schreiben.
  Je nach Kommunikationspartner sind die einzutragenden Daten von dem zu verwendenden Protokoll abhängig. Es können auch beliebige Zeichen ohne Start- u. Endezeichen eingetragen werden.

- Die Anzahl der Zeichen die gesendet werden sollen, müssen in dem Datenfeld DEC[STRX_ANZT] eingetragen werden.

- Zum Senden wird dann das SendenFlag FLG[FLGX_OUT] gesetzt. Dieses wird vom System zurückgesetzt nachdem der String gesendet wurde.

Empfangen von Daten

- Den Rahmen der Sequenz festlegen.
  Dabei werden die Anzahl und das Format der Start- u. Stopzeichen in den Datenfelder DEC[STRX_SEQ1A] und DEC[STRX_SEQ1E] festgelegt.

- Zu empfangende Sequenz mit Start und Stopzeichen definieren
  Dabei werden im Datenfeld SEQ[SEQX_A11], SEQ[SEQX_A12], SEQ[SEQX_E11] und SEQ[SEQX_E12] die Zeichen der Start- und Stopsequenz definiert.

- die Sequenz im Datenfeld FLG[FLGX_ENS1] freischalten

- die Erkennung, ob Daten angekommen sind erfolgt über das Stop-Flag im Datenfeld FLG[FLGX_STO].
  Ist dieses gesetzt ist eine Sequenz angekommen.
  Welche Sequenz erkannt wurde kann im Datenfeld DEC[STRX_SEQN] gelesen werden.
  Die Anzahl der empfangenen Zeichen kann aus dem Datenfeld DEC[STRX_ANZR] gelesen werden.

- Um wieder neue Daten empfangen zu können müssen die Flags FLG[FLGX_STA], FLG[FLGX_STO], FLG[FLGX_TIM] und FLG[FLGX_LNG] gelöscht werden.

3.12 Fehlererkennung

Nach der Initialisierung sollte der Status im Datenfeld COM_STAT überprüft werden.

COM_STAT	0	bedeutet die Schnittstelle konnte korrekt initialisiert werden.
	1	bedeutet die Schnittstelle konnte nicht initialisiert werden.

Nach dem Empfangen von Daten kann in den Datenfelder FLG[FLGX_TIM], FLG[FLGX_LNG] und FLG[FLGX_ERR] der Zustand des Funktionsblocks STRING-X ermittelt werden.

FLG[FLGX_TIM]	Timeout erkannt
FLG[FLGX_LNG]	Länge erreicht erkannt
FLG[FLGX_ERR]	Fehler beim Empfang oder der Schnittstelleninit.

3.13 Kommunikation mit StringX

3.13.1 Beschreibung der Funktionen

3.13.1.1 INIT_STRX

Funktion: Initialisierung einer Instanz

Mit dieser Funktion wird einer Instanz eine Schnittstelle Kanal 0,1,2 zugeordnet und initialisiert. Die Daten der angelegten Instanz müssen zuvor definiert sein. Hier wird die Default-Initialisierung angeboten.

Daten die nicht die Schnittstellendefinition (Baud,Parity,..)
also den MODE betreffen können geändert werden.

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Kanal	INT	0,1,2

ACHTUNG:
Die Funktion INIT_STRX verknüpft die String-Funktionalität mit dem Betriebssystem. Aus diesem Grund darf die Funktion für jede Instanz (S0:STRX oder Test:STRX) nur einmal ausgeführt werden.

3.13.1.2 SWR_STBOX

Funktion: Einen ASCII-String in die T-Box schreiben

Mit dieser Funktion kann die Sende-Box (T-BOX) aus einem oder mehreren Strings aufgebaut werden.

Bei Lng > 0 wird genau die vorgegebene Anzahl Zeichen in die TBOX geschrieben.

Bei Lng = 0 wird bis zum String-Ende-Zeichen 0 oder dem Erreichen der Maximallänge von 255 Zeichen in die TBOX geschrieben.

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	max. 16 Strings
Lng	INT	0..Länge T-Box

3.13.1.3 SRD_SRBOX

Funktion: Einen ASCII-String aus der R-Box lesen

Mit dieser Funktion können aus der Empfanges-Box (R-BOX) Strings mit definierbarer Länge gelesen, d.h in ein vorgegebenes Feld geschrieben werden.
Der String kann ab jeder gewünschten Position gelesen werden. Soll die komplette R-BOX gelesen werden wird Index auf 0 gesetzt. Sollen nur "Daten" (ohne Startzeichen) gelesen werden muß der Index auf 1 oder 2 gesetz werden.

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	max. 16 Strings
Lng	INT	0..Länge der R-Box

3.13.1.4 SWR_TBCC

Funktion: BCC-Prüfung über einen definierbaren Bereich der T-BOX (Sende-Box)

Mit dieser Funktion kann der Bereich der BCC-Prüfung der Sende-Box (T-BOX) festgelegt werden.
Mit dem Setzen des BCC-Flags wird die BCC-Prüfung gestartet. Ist die BCC-Prüfung beendet wird das BCC-Flag vom System gelöscht und das Ergebnis wird im DEC-Bereich eingetragen.
Die BCC-Prüfung bezieht sich immer auf die zuletzt ausgeführte Bereichsfestlegung (SWR_TBCC oder SWR_RBCC)

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	max. 16 Strings
Lng	INT	0..Länge T-Box

Die Funktion trägt das Berechnungsergebnis im Datenfeld DEC[STRX_BCC] ein.

3.13.1.5 SWR_RBCC

Funktion: BCC-Prüfung über einen definierbaren Bereich der R-BOX (Empfangs-Box)

Mit dieser Funktion kann der Bereich der BCC-Prüfung der Empfangs-Box (R-BOX) festgelegt werden.
Mit dem Setzen des BCC-Flags wird die BCC-Prüfung gestartet. Ist die BCC-Prüfung beendet wird das BCC-Flag vom System gelöscht und das Ergebnis wird im DEC-Bereich eingetragen.
Die BCC-Prüfung bezieht sich immer auf die zuletzt ausgeführte Bereichsfestlegung (SWR_TBCC oder SWR_RBCC)

Parametertabelle:

Parameter	Typ	Werte
String_Adr	adresse	
Instanz	DINT	
Index	INT	max. 16 Strings
Lng	INT	0..Länge R-Box

Die Funktion trägt das Berechnungsergebnis im Datenfeld DEC[STRX_BCC] ein.

Beispiel-Programm:
Mit der EMD-Taste F1 wird ein String gesendet und empfangene Daten werden immer angezeigt.

HINWEIS:
Für dieses Beispiel muß die MMI.LIB geladen werden.

3.13.2 Beispielprogramm String

3.14 Serielle Kommunikation

3.14.1 Allgemeines

Die BIOS-Funktionen der Funktionsgruppe „Serielle Kommunikation“ stellen dem Anwender Funktionen zur asynchronen Übertragung von Daten über eine serielle Schnittstelle (RS232 bzw. RS485) zur Verfügung. Durch die BIOS-Funktionen wird dabei kein festes Protokoll vorgegeben. Der Anwender kann ein beliebiges Protokoll mit diesen Funktionen aufbauen.

HINWEIS:
Bevor die Funktionen der Funktionsgruppe „Serielle Kommunikation“ benutzt werden können, muß mit der Funktion „BCom_init“ die Serielle Schnittstelle initialisiert werden.

3.14.2 Bcom Funktionsbeschreibung

3.14.2.1 BCom_init

Funktion: Initialisierung der SIO und FIFO's (Input und Output)

Syntax:

```
#include "9001.h"

int BCom_init( unsigned int kanal,
      unsigned int mode )
```

Parameter:

unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

unsigned int mode: Schnittstellenparameter
Bit 0: Freigabe Parity-Ueberwachung
define:
PARITY_ERROR_ENA
Bit 1: Freigabe Frame-Ueberwachung
define:
FRAME_ERROR_ENA
Bit 2: Freigabe Overrun-Ueberwachung
define:
OVERRUN_ERROR_ENA
Bit 3: Umschaltung Voll-Halbduplex
define:
VOLLDUPLEX,
HALBDUPLEX
Bit 4: Anzahl Stopbits
define:
_1STOPBIT,
_2STOPBIT
Bit 5..7: Betriebsart
define:
_7BIT_PARITYEVEN,
_7BIT_PARITYODD,
_8BIT_NOPARITY,
_8BIT_PARITYEVEN,
_8BIT_PARITYODD
Bit 8..11: Uebertragungsgeschwindigkeit
define:
_1200BAUD,
_2400BAUD,
_4800BAUD,
_9600BAUD,
_19200BAUD,
_38400BAUD,
_57600BAUD
Bit 12..15: Timeoutueberwachung Input
define:
NO_TIMEOUT,
TIMEOUT_10MS,
TIMEOUT_20MS,
TIMEOUT_30MS,

TIMEOUT_40MS,
TIMEOUT_50MS,
TIMEOUT_75MS,
TIMEOUT_100MS,
TIMEOUT_150MS,
TIMEOUT_200MS,
TIMEOUT_250MS,
TIMEOUT_300MS
TIMEOUT_400MS,
TIMEOUT_500MS,
TIMEOUT_750MS,
TIMEOUT_1000MS

Beschreibung: Initialisierung der SIO entsprechend der Uebergabeparameter. Die definierten Konstanten können mit dem Oder-Operator (|) verknüpft werden.

Die Schnittstelle ASC2 kann mit Handshake-Signalen RTS und CTS betrieben werden. Sollen diese Leitungen benutzt werden, dann muss die Schnittstelle im Vollduplex-Modus initialisiert werden.

Rückgabewert: int ret_code:
define: BIOS_OK (Wert 0) = ok
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal oder Mode)

siehe auch: -----

Beispiele: Initialisierung der internen SIO mit folgenden Parametern: Baudrate: 9600, 7 Datenbits, Parity even, 1 Stopbit, Halbduplex, Überprüfung Parityerror, Überprüfung Frameerror, Überprüfung Overrunerror, Timeout Input vom 150 ms.

```
#include "9001.h“

void main (void)
{
  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
  for(;;)
  {
    BCom_send(ASC0, "ABCDE“, 3);
  }
         }
```

3.14.2.2 BCom_getcts

Funktion: Zustand CTS-Leitung lesen.

Syntax:
```
#include "9001.h“

unsigned int BCom_getcts(unsigned int kanal )
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

Beschreibung: Liefert Zustand CTS-Leitung

Rückgabewert: int ret_code
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal)
1 wenn CTS-Signal aktiv (positive Spannung an CTS)
0 wenn CTS-Signal inaktiv (negative Spannung an CTS)

siehe auch: BCom_setrts

Beispiele: CTS-Signal von ASC2 als Handshake zum Senden verwenden.

```
#include "9001.h“

void main (void)
{

  /* Init. Schnittstelle */
  BCom_init(ASC2, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | VOLLDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
.
.
.

  /* RTS-Signal aktivieren */
  BCom_setrts(ASC2,1);

  /* Auf CTS-Signal warten */
  while (BCom_getcts(ASC2) != 1)
  {
  ;
  }

  /* Sende-Vorgang starten */
  BCom_send (ASC2, &sende_puffer, anzahl);


  /* CTS-Signal überwachen bis Sende-FIFO leer */
  while((BCom_stat(ASC2) & OUTP_FIFO_EMPTY) !=
          OUTP_FIFO_EMPTY)
  {
     if (BCom_getcts(ASC2) == 1)
     {
        BCom_sendenable(ASC2);
     }

     else
     {
        BCom_senddisable(ASC2);
     }
  }

  /* RTS-Signal deaktivieren */
  BCom_setrts(ASC2,0);

.
.
.
}
```

3.14.2.3 BCom_setrts

Funktion: Zustand RTS-Leitung setzen.

Syntax:

```
#include "9001.h“

unsigned int BCom_setrts(unsigned int kanal,
unsigned int zustand)
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

unsigned int zustand: 0, RTS inaktiv
1, RTS aktiv

Beschreibung: Setzt Zustand RTS-Leitung

Rückgabewert: int ret_code
define: BIOS_OK
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal)
Wert 1 = Falscher Übergabewert

siehe auch: BCom_getcts, BCom_init

Beispiele: RTS-Signal von ASC2 als Handshake zum Senden verwenden.

```
#include "9001.h“

void main (void)
{

  /* Init. Schnittstelle */
  BCom_init(ASC2, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | VOLLDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
.
.
.

  /* RTS-Signal aktivieren */
  BCom_setrts(ASC2,1);

  /* Auf CTS-Signal warten */
  while (BCom_getcts(ASC2) != 1)
  {
  ;
  }

  /* Sende-Vorgang starten */
  BCom_send (ASC2, &sende_puffer, anzahl);


  /* CTS-Signal überwachen bis Sende-FIFO leer */
  while((BCom_stat(ASC2) & OUTP_FIFO_EMPTY) !=
          OUTP_FIFO_EMPTY)
  {
     if (BCom_getcts(ASC2) == 1)
     {
        BCom_sendenable(ASC2);
     }

     else
     {
        BCom_senddisable(ASC2);
     }
  }

  /* RTS-Signal deaktivieren */
  BCom_setrts(ASC2,0);

.
.
.
}
```

3.14.2.4 BCom_send

Funktion: Eintrag von einem oder mehreren Zeichen in die Output- FIFO.

Syntax:

```
#include "9001.h"

int BCom_send(unsigned int kanal,
      unsigned char *quell_ptr,
      unsigned int anzahl )
```

Parameter:
unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2
unsigned char *quell_ptr: Zeiger auf Quellspeicher
unsigned int anzahl: Anzahl Zeichen die ausgegeben werden.

Beschreibung: In die Output-FIFO werden entsprechend der Anzahl die übergebenen Zeichen eingetragen. Können nicht alle Zeichen in die Output-FIFO eingetragen werden, wartet die Funktion bis das letzte Zeichen eingetragen werden kann.
Wenn die Ausgabe von der Funktion BCom_senddisablegesperrt ist, werden keine weiteren Zeichen in die Output-FIFO eingetragen.

Rückgabewert:
int ret_code:
define: BIOS_OK (Wert 0) = ok
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal)

siehe auch: -----

Beispiele: Ständige Ausgabe des Zeichenstring „ABCDE“.

```
#include "9001.h“

void main (void)
{
  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
OVERRUN_ERROR_ENA | TIMEOUT_150MS);
  for(;;)
  {
    BCom_send(ASC0, "ABCDE“, 5);
  }
}
```

3.14.2.5 BCom_senddirect

Funktion: Direkte Ausgabe eines Zeichen mit Umgehung der Output-FIFO.

Syntax:
```
#include "9001.h“

int BCom_senddirect(unsigned int kanal,
unsigned int zeichen)
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2
unsigned int zeichen: Ausgabezeichen

Beschreibung: Direkte Ausgabe eines Zeichen mit Umgehung des Output-FIFO. Pro Funktionsaufruf kann maximal ein Zeichen an der Output-FIFO vorbei übertragen werden. Die Ausgabe erfolgt als nächstes Zeichen.

Rückgabewert: int ret_code:
define: BIOS_OK (Wert 0) = ok
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal, Ausgabespeicher belegt)

siehe auch: -----

Beispiele: Ständige direkte Ausgabe des Zeichen „S“. Die Ausgabe über die Output-FIFO wurde disabled.

```
#include "9001.h“

void main (void)
{
  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
OVERRUN_ERROR_ENA | TIMEOUT_150MS);

  BCom_senddisable(ASC0);

  for(;;)
  {
    BCom_senddirect(ASC0, ‘S’);
    BCom_send(ASC0, "EPIS“, 4);
  }
}
```

3.14.2.6 BCom_senddisable

Funktion: Die Zeichenausgabe über die Output-FIFO wird unterbunden.

Syntax:
```
#include "9001.h“

int BCom_senddisable(unsigned int kanal)
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

Beschreibung: Die Zeichenausgabe über die Output-FIFO wird unterbunden. Der Inhalt der Output-FIFO-Inhalt bleibt erhalten. Mit der Funktion „BCom_senddirect“ kann die Ausgabe trotz Unterbindung fortgesetzt werden.

Rückgabewert: int ret_code:
define: BIOS_OK (Wert 0) = ok
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal, FIFO bereits disabled)

siehe auch: -----

Beispiele: Ausgabe Output-FIFO unterbinden (disable).
Der Eintrag des Zeichenstring in die Output-FIFO ist nicht möglich (disabled). Nach der direkten Ausgabe des Zeichen „S“ wird die Ausgabe über die Output-FIFO wieder enabled.

```
#include "9001.h“

void main (void)
{
  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);

  BCom_senddisable(ASC0);
  BCom_send(ASC0, „ABCDE“, 5);

  BCom_senddirect(ASC0, ‘S’);
  BCom_sendenable(ASC0);
}
```

3.14.2.7 BCom_sendenable

Funktion: Freigabe der Ausgabe (Output-FIFO).

Syntax:
```
#include "9001.h“

int BCom_sendenable(unsigned int kanal)
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

Beschreibung: Freigabe der Zeichenübertragung aus der Output-FIFO.

Rückgabewert: int ret_code:
define: BIOS_OK (Wert 0) = ok
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal, FIFO bereits enabled)

siehe auch: -----

Beispiele: Ausgabe Output-FIFO unterbinden (disable).
Kein Eintrag des Zeichenstring in die Output-FIFO.
Nach der direkten Ausgabe des Zeichen „S“ wird die Ausgabe über die Output-FIFO wieder enabled.

```
#include "9001.h“

void main (void)
{
  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);

  BCom_senddisable(ASC0);
  BCom_send(ASC0, „ABCDE“, 5);

  BCom_senddirect(ASC0, ‘S’);
  BCom_sendenable(ASC0);
}
```

3.14.2.8 BCom_receive

Funktion: Auslesen von Zeichen aus der Input-FIFO

Syntax:

```
#include "9001.h“

int BCom_receive(unsigned int kanal,
      unsigned char *ziel_ptr,
      unsigned int max_anzahl,
      unsigned int fifo_flag )
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2
unsigned char *ziel_ptr: Zeiger auf Zielspeicher
unsigned int max_anzahl: Maximal Anzahl Zeichen die gelesen werden soll
unsigned int fifo_flag:
0 -> Der Empfangs-Fifopointer wird entspechend der Anzahl Zeichen angepasst
1 -> Der Empfangs-Fifopointer bleibt unveraendert

Beschreibung: Auslesen der Input-FIFO.

Rückgabewert: int anzahl:
0...Fifolänge = Anzahl Zeichen die aus der Input-FIFO gelesen wurden.
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal)

siehe auch: -----

Beispiele: Zyklisches Auslesen eines Zeichen aus der Input-FIFO.

```
#include "9001.h“

void main (void)
{
  unsigned char zeichen;

  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
  while(1)
  {
    BCom_receive(ASC0, zeichen, 1, 0);
  }
}
```

3.14.2.9 BCom_checkfifo

Funktion: Überprüfung der FIFO's (Input oder Output)

Syntax:

```
#include "9001.h“

int BCom_checkfifo(unsigned int kanal,
unsigned int fifo )
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

unsigned int fifo: define: INPUT_FIFO, OUTPUT_FIFO

Beschreibung: Überprüfung der FIFO's. Bei der Input-FIFO wird die Anzahl der eingetragenen Zeichen zurückgegeben. Bei der Output-FIFO wird die Anzahl noch freier Eintragung zurückgegeben.

Rückgabewert: int ret_code:
Bei Anwahl Input: Anzahl Zeichen die in der Input-FIFO enthalten sind
Bei Anwahl Output: Anzahl Zeichen die noch in die Output-FIFO eingetragen werden koennen.
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal, falsche FIFO-Anwahl)

siehe auch: -----

Beispiele: Auslesen eines Zeichen aus der Input-FIFO.

```
#include "9001.h“

void main (void)
{
  unsigned char test_string[100];
  unsigned int anzahl;

  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
  while(1)
  {
    anzahl = BCom_checkfifo(ASC0, INPUT_FIFO);
    if (anzahl > 0)
      BCom_receive(ASC0, test_string, anzahl, 0);
  }
}
```

3.14.2.10 BCom_clearfifo

Funktion: Löschen der FIFO's (Input oder Output oder beide)

Syntax:

```
#include "9001.h“

int Bcom_clearfifo(unsigned int kanal,
unsigned int fifo)
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2
unsigned int fifo: define: INPUT_FIFO, OUTPUT_FIFO
oder beide FIFO's

Beschreibung: Rücksetzen der FIFO's.

Rückgabewert: int ret_code:
define: BIOS_OK (Wert 0) = ok
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal, falsche FIFO-Anwahl)

siehe auch: -----

Beispiele: Check des Schnittstellen-Status, ob Übertragungsfehler vorliegen.

```
#include "9001.h“

void main (void)
{
  unsigned char test_string[100];
  int anzahl, status;
  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
  while(1)
  {
    status = BCom_stat(ASC0);
    /* Check ob Übertragungsfehler vorliegt */
    if (status & 0x03)
    {
      BCom_clearfifo(ASC0, INPUT_FIFO);
      BCom_clearstat(ASC0);
    }
    anzahl = BCom_checkfifo(ASC0, INPUT_FIFO);
    if (anzahl > 0)
      BCom_receive(ASC0, test_string, anzahl, 0);
  }
        }
```

3.14.2.11 BCom_stat

Funktion: Überprüfung des SIO-Status und der FIFO's

Syntax:
```
#include "9001.h“

int BCom_stat(unsigned int kanal )
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

Beschreibung: Überprüfung des SIO-Status und der FIFO's

Rückgabewert: int ret_code:
Bit0: 1 = Parity-Error
Bit1: 1 = Frame-Error
Bit2: 1 = Overrun-Error
Bit3: 1 = Overlow Input-Fifo
Bit4: 1 = Timeout abgelaufen
Bit5: 1 = Input-Fifo empty (leer)
Bit6: 1 = Output-Fifo empty (leer)
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal)

siehe auch: -----

Beispiele: Überprüfung nach Übertragungsfehlern

```
#include "9001.h“

void main (void)
{
  unsigned char test_string[100];
  int anzahl, status;
  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
  while(1)
  {
    status = BCom_stat(ASC0);
    /* Check ob Übertragungsfehler vorliegt */
    if (status & 0x03)
    {
      BCom_clearfifo(ASC0, INPUT_FIFO);
      BCom_clearstat(ASC0);
    }
    anzahl = BCom_checkfifo(ASC0, INPUT_FIFO);
    if (anzahl > 0)
      BCom_reseive(ASC0, test_string, anzahl, 0);
  }
            }
```

3.14.2.12 BCom_clearstat

Funktion: Löschen des SIO-Status.

Syntax:
```
#include "9001.h“

unsigned int BCom_clearstat(unsigned int kanal )
```

Parameter: unsigned int kanal: Kanalnummer
define: ASC0 -> interne Schnittstelle
define: ASC1 -> externe Schnittstelle 1
define: ASC2 -> externe Schnittstelle 2

Beschreibung: Löschung des Schnittstellenstatus (SIO und FIFO-Fehler)

Rückgabewert: int ret_code:
define: BIOS_OK (Wert 0) = ok
define: BIOS_ERROR (Wert -1) = Fehler (falscher Kanal, falsche FIFO-Anwahl)

siehe auch: -----

Beispiele: Check des Schnittstellen-Status, ob Übertragungsfehler vorliegen.

```
#include "9001.h“

void main (void)
{
  unsigned char test_string[100];
  int anzahl, status;

  /* Init. Schnittstelle */
  BCom_init(ASC0, _9600BAUD | _7BIT_PARITYEVEN |
            1STOPBIT | HALBDUPLEX |
            PARITY_ERROR_ENA | FRAME_ERROR_ENA |
            OVERRUN_ERROR_ENA | TIMEOUT_150MS);
  while(1)
  {
    status = BCom_stat(ASC0);
    /* Check ob Übertragungsfehler vorliegt */
    if (status & 0x03)
    {
      BCom_clearfifo(ASC0, INPUT_FIFO);
      BCom_clearstat(ASC0);
    }
    anzahl = BCom_checkfifo(ASC0, INPUT_FIFO);
    if (anzahl > 0)
    {
      BCom_reseive(ASC0, test_string, anzahl, 0);
    }
  }
}
```