

SIO4/8

Four/Eight Channel High Speed Serial I/O

**All SIO4 and SIO8 Models
All Form Factors
All Standard Zilog Versions**

HDLC Protocol Library Reference Manual

Manual Revision: March 19, 2024

**General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: <http://www.generalstandards.com>
E-mail: sales@generalstandards.com
E-mail: support@generalstandards.com**

Preface

Copyright © 2013-2024, **General Standards Corporation**

Additional copies of this manual or other literature may be obtained from:

General Standards Corporation

8302A Whitesburg Dr.

Huntsville, Alabama 35802

Phone: (256) 880-8787

FAX: (256) 880-8788

URL: <http://www.generalstandards.com>

E-mail: sales@generalstandards.com

General Standards Corporation makes no warranty of any kind with regard to this documentation and/or software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release, **General Standards Corporation** assumes no responsibility for any errors, inaccuracies or omissions herein. This documentation, information and software are made available solely on an “as-is” basis. Nor is there any commitment to update or keep current this documentation.

General Standards Corporation does not assume any liability arising out of the application or use of documentation, software, product or circuit described herein, nor is any license conveyed under any patent rights or any rights of others.

General Standards Corporation assumes no responsibility for any consequences resulting from omissions or errors in this manual or from the use of information contained herein.

General Standards Corporation reserves the right to make any changes, without notice, to this documentation, software or product, to improve accuracy, clarity, reliability, performance, function, or design.

ALL RIGHTS RESERVED.

GSC is a trademark of **General Standards Corporation**.

PLX and PLX Technology are trademarks of PLX Technology, Inc.

Zilog and Z16C30 are trademarks of Zilog, Inc.

Table of Contents

1. Introduction	7
1.1. Purpose	7
1.2. Acronyms	7
1.3. Definitions	7
1.4. Software Overview.....	7
1.5. Hardware Overview	7
1.6. Exclusions.....	8
1.6.1. Global Rx FIFO Full Configuration	8
1.7. Reference Material.....	8
2. The HDLC Serial Protocol	9
2.1. Description	9
2.2. History.....	9
3. Library Interface Files	11
3.1. Header File	11
3.2. Static Library Files.....	11
4. Library Interface	12
4.1. Functions.....	12
4.1.1. sio4_hdlc_close().....	12
4.1.2. sio4_hdlc_get()	12
4.1.3. sio4_hdlc_init()	12
4.1.4. sio4_hdlc_init_data()	13
4.1.5. sio4_hdlc_ioctl().....	14
4.1.6. sio4_hdlc_open()	15
4.1.7. sio4_hdlc_rx_flush()	16
4.1.8. sio4_hdlc_rx_frame().....	16

4.1.9. sio4_hdlc_set()	19
4.1.10. sio4_hdlc_show()	20
4.1.11. sio4_hdlc_tx_abort()	20
4.1.12. sio4_hdlc_tx_break()	21
4.1.13. sio4_hdlc_tx_flush()	21
4.1.14. sio4_hdlc_tx_frame()	22
4.1.15. sio4_hdlc_tx_status()	24
4.1.16. sio4_hdlc_tx_wait()	25
4.2. Data Structures	26
4.2.1. sio4_hdlc_t	26
5. Operating Information	50
5.1. Basic Illustration	50
5.2. Getting Started	50
5.2.1. Cable Validation	50
5.2.2. Customizing the Configuration	51
5.3. Debugging Aids	51
5.3.1. Device Identification	52
5.3.2. sio4_hdlc_show()	52
5.3.3. Detailed Register Dump	52
5.3.4. Status Return Values	53
5.4. Tx Abort (Transmitter Abort Sequence)	53
5.5. Tx Preamble (Transmitter Frame Preamble)	53
5.6. Tx CRC (Transmitter Frame Check Sequence)	53
5.7. Rx CRC (Receiver Frame Check Sequence)	54
5.8. Basic Transmission Requirements	54
5.8.1. Send Data as a Single Complete Frame	54
5.8.2. Tx Short Frame Status and Preload	55

5.8.3. Complete Frame Transmission Before Returning	55
5.8.4. Transmit Frames ASAP	56
5.9. Clocking Configurations	56
5.9.1. Four Signal Configuration: Cable Tx Clock is Used, Cable Rx Clock is Used	57
5.9.2. Three Signal Configuration #1: Cable Tx Clock is Unused, Cable Rx Clock is Used	58
5.9.3. Three Signal Configuration #2: Cable Tx Clock is Used, Cable Rx Clock is Unused	59
5.9.4. Two Signal Configuration: Cable Tx Clock is Unused, Cable Rx Clock is Unused	60
5.10. Cable Configuration Modes	61
5.10.1. DCE/DTE Mode	61
5.10.2. Legacy Mode	61
5.11. Error and Status Detection	61
5.11.1. Interrupt Events	61
5.11.2. Rx Status Word	62
Document History	63

Table of Figures

Figure 1 The HDLC Frame format.	9
Figure 2 A functional illustration of an SIO4B or later model board.	50
Figure 3 The clock routing produced when the Cable Tx Clock is used and the Cable Rx Clock is used.	57
Figure 4 The clock routing produced when the Cable Tx Clock is unused and the Cable Rx Clock is used.	58
Figure 5 The clock routing produced when the Cable Tx Clock is used and the Cable Rx Clock is unused.	59
Figure 6 The clock routing produced when the signals Cable Tx Clock and Cable Rx Clock are unused.	60

1. Introduction

1.1. Purpose

This purpose of this document is to give a complete description of the HDLC Protocol Library programming interface.

1.2. Acronyms

The following is a list of commonly occurring acronyms used throughout this document.

Acronyms	Description
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DPLL	Digital Phase Lock Loop
FCS	Frame Check Sequence
GSC	General Standards Corporation
HDLC	High-level Data Link Control
PCI	Peripheral Component Interconnect
PMC	PCI Mezzanine Card
RCC	Receive Character Counter
USC	Universal Serial Controller

1.3. Definitions

The following is a list of commonly occurring terms used throughout this document.

Term	Definition
Application	Application means the user mode process, which runs in user space with user mode privileges.
Driver	Driver means the executable providing the direct access to the SIO4 hardware.
Frame	This term refers to a block of data, with overhead, sent using the HDLC protocol.
Library	Depending on context, this is a general reference to the HDLC Protocol Library.
RCC FIFO	This is an internal USC FIFO that records the length of received frames.
SIO4	This is used as a general reference to any Zilog based board supported by this driver. This includes both SIO4 and SIO8 model boards. It is also used to refer to revisions of the board that do not include a suffix following the '4', such as SIO4A or SIO4B.

1.4. Software Overview

The HDLC Protocol Library is a statically linked library providing an HDLC centric interface to the SIO4 device driver. The library is provided in source form and must be built before being used. The library is a thin software layer that sits between an SIO4 application and the SIO4 API Library. The interface provided by the library is HDLC specific and is a simplified rendition of the IOCTL services that are part of the overall driver interface. The library exists in parallel with the driver interface.

1.5. Hardware Overview

NOTE: The SIO8 boards appear to the system as two SIO4 boards.

The SIO4 is a four channel high-speed serial interface I/O board. This board provides for bi-directional serial data transfers between two computers, or one computer and an external peripheral. Once the data link between the two devices is established, the desired transfers can be performed and will become transparent to the user. The SIO4 board includes two DMA controllers and comes with a maximum of 256K Bytes of FIFO storage, which is 32K per

channel direction (32K * 2 * 4). Each DMA controller is capable of transferring data to and from host memory; whereas the FIFO help maintain continuous data transfer at the cable interface. The FIFO configuration can vary greatly from one SIO4 version to another (i.e. 32K * 2 * 4 to 1K * 2 * 1 to none at all). The SIO4 comes with transceivers that are fixed as RS232 or RS485/422, or with transceivers that are configurable. The SIO4 comes in two basic varieties; SYNC models or Zilog models, which are based on two Z16C30 dual USC chips. Later model SIO4 boards support both models with the mode being software controlled on a per channel basis. The SIO4 also provides for interrupt generation for various states of the board like Sync Character detection, FIFO empty, FIFO full and DMA complete.

NOTE: Software selection of SYNC or Zilog mode of operation is not at this time explicitly supported by the Protocol Libraries, the SIO4 API Library or the device driver. The operating mode is controlled by the model ordered.

1.6. Exclusions

1.6.1. Global Rx FIFO Full Configuration

The global Rx FIFO Full Configuration setting (see `SIO4_IOCTL_RX_FIFO_FULL_CFG_GLB` in `sio4.h`) is not included as part of the HDLC Protocol Library. It is excluded because the setting overrides the channel specific settings for all four channels.

1.7. Reference Material

The following reference material may be of particular benefit in using the SIO4 and this driver. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- ISO/IEC 13239, Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures
- The applicable *SIO4/SIO8 User Manual* from General Standards Corporation.
- The applicable *SIO4/SIO8 Driver User Manual* from General Standards Corporation.
- The *PCI Bus Master Interface Chip* data handbook for the PCI9056 or PCI9080 from PLX Technology, Inc.

PLX Technology Inc.
870 Maude Avenue
Sunnyvale, California 94085 USA
Phone: 1-800-759-3735
WEB: <http://www.plxtech.com/>

- The *Z16C30 USC User's Manual* from Zilog.

Zilog, Inc.
910 E Hamilton Ave
Campbell, California 95008 USA
Phone: 1-408-558-8500
WEB: <http://www.zilog.com/>

2. The HDLC Serial Protocol

2.1. Description

HDLC is a bit oriented serial transmission protocol centered about what is called the Flag. The Flag is a series of eight bits whose values are 01111110. The Flag is used to signal the beginning and end of a Frame. Refer to Figure 1 below for the basic layout of an HDLC Frame. The only other special bit sequences are the Aborts, which consists of a zero followed by seven or more ones, and the Break, which is an Abort that is typically many milliseconds in duration. When sending out other than these special sequences, the transmitter performs bit stuffing by converting any 0111111 sequence into 0111101. The receiver performs the reverse operation by converting the sequence 0111101 to 0111111. In this way, the HDLC protocol places no restriction on the content of the serial data.

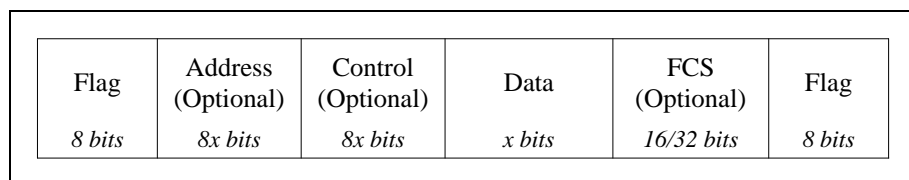


Figure 1 The HDLC Frame format.

Within a Frame, everything but the data is always in multiples of eight bits. The Flags are always eight bits wide. All Address characters, Control characters and FCS characters are always in multiples of eight bits. The data though, when present, is in multiples of from one to eight bits. All data characters within a Frame are the same size, except for the last data character, which may be smaller.

The Address field may be a fixed size or a variable size. When variable, it contains a fixed sized portion that is conditionally followed by additional Address bytes. The last byte of the fixed size portion and every byte thereafter is checked to see if the following byte is part of the Address field. The check examines the least significant bit of the byte. (The hardware checks the first bit received even if software has configured the device for Most Significant Bit First reception.) If the least significant bit is clear, then the next byte is included as part of the variable sized Address field. Though the Address field may contain a number of bytes, only the first byte is used by the SIO4 hardware for address comparison. If the first byte is all ones or if the value matches a preset address value, then the Frame is captured. Otherwise the Frame is ignored.

The captured Control field may also be a fixed size or a variable size. When variable, it contains a fixed sized portion that is conditionally followed by additional Control bytes. The last byte of the fixed size portion and every byte thereafter is checked to see if the following byte is part of the Control field. The check examines the most significant bit of the byte. (The hardware checks the last bit received even if software has configured the device for Most Significant Bit First reception.) If the most significant bit is set, then the next byte is included as part of the variable sized Control field. After this, one additional byte is included in the Control field.

The FCS, or Frame Check Sequence, is a CRC calculated over the content of frame from the first Address byte to the last Data byte. The SIO4 can generate CRCs of 16 or 32 bits. The FCS is inserted automatically by the transmitter as the frame is being sent. On reception, the calculated FCS is compared against the received FCS. The result of the comparison is included in the frame status.

While a frame is not being transmitted, the transmitter is typically configured to output a continuous stream of Flag sequences. These are repeated until the next Frame is sent. At the very minimum, individual frames may be separated by only a single Flag sequence. This results in the sequence FCS-Flag-Address. The SIO4 transmitter can also be configured to create a minimum inter-frame delay consisting of up to 10 Flag sequences.

2.2. History

The current official HDLC specification is documented by ISO 13239. This was preceded by a number of other ISO specifications. There are also a number of additional specifications for various HDLC subsets, derivatives and other

aspects of implementation. The origin for HDLC is SDLC, which was designed by IBM in 1975 for use with its SNA traffic. SDLC was subsequently submitted by IBM for codification. After some modifications it was renamed HDLC and was standardized by ISO 3309, which was eventually replaced by ISO 13239.

3. Library Interface Files

This section gives general information on the HDLC Protocol Library interface files.

3.1. Header File

The library's interface is defined via the header file shown below. To use the HDLC Protocol Library applications must include this header file in their sources. Including this header file pulls in all other pertinent SIO4 specific header files. Therefore, sources may include only this one SIO4 header and make files may reference only this one SIO4 include directory.

File	Location
sio4_hdlc.h	.../sio4/include

3.2. Static Library Files

The executable code for the API defined for the HDLC Protocol Library is contained in the static library file `sio4_hdlc.a`, which is identified below. Using this library however, requires linking in other SIO4 specific static libraries. For this reason, and for ease of use, it is recommended that application make files link in the SIO4 Main Library instead of the HDLC static library along with all of its dependencies. The result is that application make files reference only a single SIO4 static library and only a single SIO4 static library path.

Library	File	Location
HDLC Protocol Library	sio4_hdlc.a	.../sio4/lib
SIO4 Main Library	sio4_main.a *	.../sio4/lib

* Refer to the SIO4 API Library Reference Manual for clarification when using multiple GSC product types in the same application.

4. Library Interface

The library interface is defined via the header file `sio4_hdlc.h`, which is located in the `.../sio4/include/` directory.

4.1. Functions

The library header defines the complete HDLC interface offered by the library, which includes the following function declarations.

4.1.1. `sio4_hdlc_close()`

This function is the entry point to close a connection previously opened to an SIO4 for HDLC operation. All resources allocated by the library for the opened device are released as part of the close operation. This includes freeing allocated memory and closing access to the SIO4.

Prototype

```
int sio4_hdlc_close(int fd);
```

Argument	Description
<code>fd</code>	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.2. `sio4_hdlc_get()`

This function retrieves the current settings from the SIO4 for each of the referenced structure's fields.

Prototype

```
int sio4_hdlc_get(int fd, sio4_hdlc_t* hdlc, const char** err);
```

Argument	Description
<code>fd</code>	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
<code>hdlc</code>	This is the structure where the settings are to be recorded. Any field pertaining to an unsupported feature will be set to -1. The value -2 indicates a setting that is invalid. (Refer to section 4.2.1, page 26.)
<code>err</code>	In the event of an error this will be set to identify the source of the error. This may be NULL.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.3. `sio4_hdlc_init()`

This function initializes the HDLC Protocol Library and must be the first call into the library.

NOTE: This function is NOT multithread safe. All other HDLC Protocol Library calls are thread safe AFTER this function completes successfully.

NOTE: This service initializes the HDLC Protocol Library as well as the SIO4 API Library.

NOTE: This function may be called more than once, but only the first successful call initializes the library. Any subsequent call has no effect.

NOTE: This function does not alter the state of the SIO4 being accessed or any of its settings. This call does not place the channel in an HDLC state and does not alter the state that the device is in.

Prototype

```
int sio4_hdlc_init(void);
```

Argument	Description
None	The function has no arguments.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.4. sio4_hdlc_init_data()

This function initializes an `sio4_hdlc_t` structure according to the capabilities of the accessed device and some basic caller preferences. The `sio4_hdlc_t` bit-rate related fields are initialized according to the content of the provided `sio4_hdlc_init_t` structure. The non-bit-rate related fields are set either to defaults consistent with HDLC operation of the channel or to defaults required to satisfy clocking signal use as specified in the `sio4_hdlc_init_t` structure. Upon return from this call `sio4_hdlc_t` fields may be modified to meet application requirements.

NOTE: This function does not alter the state of the SIO4 being accessed or any of its settings. This call does not place the channel in an HDLC state and does not alter the state that the device is in.

Prototype

```
int sio4_hdlc_init_data(
    int          fd,
    const sio4_hdlc_init_t* init,
    sio4_hdlc_t*  hdlc,
    const char**  err);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
init	This structure provides the basic information needed to initialize numerous fields in the next structure. See below for more information.
hdlc	This is the structure that the call will initialize. Any field pertaining to an unsupported feature will be set to -1. (Refer to section 4.2.1, page 26.)
err	In the event of an error this will be set to identify the source of the error. This may be NULL.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

Data Type

This structure contains information used to configure HDLC clocking for the USC transmitter and receiver.

NOTE: For additional information refer to Clocking Configuration (section 5.9, page 56), which gives information on initializing this structure.

```
typedef struct
{
    s32      tx_bit_rate;
    s32      rx_bit_rate;
    s32      cbl_txc;
    s32      cbl_rxc;
    s32      osc_prog;
} sio4_hdlc_init_t;
```

Field	Description						
tx_bit_rate	This is the desired bit rate for the transmitter. This value must be greater than or equal to one, and less than or equal to 20,000,000.						
rx_bit_rate	This is the desired bit rate for the receiver. This value must be greater than or equal to one, and less than or equal to 20,000,000.						
cbl_txc	<div> <p>This field is used to indicate if the Cable Tx Clock signal is required to output the transmit clock. Valid values are given in the table below.</p> <table> <tr> <th>Values</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_CBL_TXC_UNUSED</td><td>The transmit clock is not routed to the Cable Tx Clock signal. The signal may be used for some other purpose.</td></tr> <tr> <td>SIO4_HDLC_CBL_TXC_USED</td><td>The transmit clock is routed to the Cable Tx Clock signal.</td></tr> </table> </div>	Values	Description	SIO4_HDLC_CBL_TXC_UNUSED	The transmit clock is not routed to the Cable Tx Clock signal. The signal may be used for some other purpose.	SIO4_HDLC_CBL_TXC_USED	The transmit clock is routed to the Cable Tx Clock signal.
Values	Description						
SIO4_HDLC_CBL_TXC_UNUSED	The transmit clock is not routed to the Cable Tx Clock signal. The signal may be used for some other purpose.						
SIO4_HDLC_CBL_TXC_USED	The transmit clock is routed to the Cable Tx Clock signal.						
cbl_rxc	<div> <p>This field is used to indicate if the receiver will receive its clock from the cable's Rx Clock signal. Valid values are given in the table below.</p> <table> <tr> <th>Values</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_CBL_RXC_UNUSED</td><td>The Cable Rx Clock signal is not used to clock data into the receiver. The signal may be used for some other purpose.</td></tr> <tr> <td>SIO4_HDLC_CBL_RXC_USED</td><td>The Cable Rx Clock signal is driven by external equipment to clock data into the receiver.</td></tr> </table> </div>	Values	Description	SIO4_HDLC_CBL_RXC_UNUSED	The Cable Rx Clock signal is not used to clock data into the receiver. The signal may be used for some other purpose.	SIO4_HDLC_CBL_RXC_USED	The Cable Rx Clock signal is driven by external equipment to clock data into the receiver.
Values	Description						
SIO4_HDLC_CBL_RXC_UNUSED	The Cable Rx Clock signal is not used to clock data into the receiver. The signal may be used for some other purpose.						
SIO4_HDLC_CBL_RXC_USED	The Cable Rx Clock signal is driven by external equipment to clock data into the receiver.						
osc_prog	This is the frequency to which the channel's on-board oscillator is to be programmed. Refer to the clocking options for additional information (section 5.9, page 56).						

4.1.5. sio4_hdlc_ioctl()

This function is the entry point to performing IOCTL operations on the device. Refer to the driver reference manual for complete information on the driver's set of IOCTL services.

Prototype

```
int sio4_hdlc_ioctl(int fd, int request, void* arg);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
request	This is an IOCTL macro contained in <code>sio4.h</code> or <code>sio4_usc.h</code> .
arg	This is the argument type required for the above referenced IOCTL service.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.6. sio4_hdlc_open()

This function is the entry point to open a connection to an SIO4 serial channel for HDLC operation. The handle returned by this call is used for all subsequent access to the specified channel. The file descriptor returned can be used for access to the library functions and the driver interface.

NOTE: Open requests will fail if the referenced device does not support the HDLC Serial Protocol. This applies to all SYNC and other non-Z16C30 DUART based boards.

NOTE: With a successful open the device is placed in an initialized, non-HDLC state. Placing the device in an HDLC state is done either through the `sio4_hdlc_set()` function (section 4.1.9, page 19) or through corresponding `sio4_hdlc_ioctl()` services (section 4.1.5, page 14).

NOTE: If the value of the index argument is specified as `-1`, then the function opens the file `/proc/sio4` for reading. In this case the `share` argument is ignored.

NOTE: If the value of the index argument is specified as `-1` then applications can retrieve information about the driver and the devices detected. Refer to the SIO4 Reference Manual for additional information. The file descriptor returned when the `-1` value is provided can be used only with `sio4_hdlc_rx_frame()` (section 4.1.8, page 16) and `sio4_hdlc_close()` (section 4.1.1, page 12). It may also be used with the SIO4 API functions `sio4_read()` and `sio4_close()` (refer to the *SIO4 Reference Manual*). The file descriptor cannot be used with any other services.

Prototype

```
int sio4_hdlc_open(int index, int share, int* fd);
```

Argument	Description						
index	This is the zero based index of the SIO4 serial channel to access.						
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).						
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table border="1"> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>>= 0</td><td>This is the handle to use to access the device in subsequent calls.</td></tr> <tr> <td>-1</td><td>There was an error. The device is not accessible.</td></tr> </table>	Value	Description	>= 0	This is the handle to use to access the device in subsequent calls.	-1	There was an error. The device is not accessible.
Value	Description						
>= 0	This is the handle to use to access the device in subsequent calls.						
-1	There was an error. The device is not accessible.						

Return Value	Description
>= 0	A valid file access handle.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.6.1. Access Modes

The value of the `share` argument determines the device access mode, as follows.

Shared Access Mode:

If the `share` argument is non-zero the device is opened in Shared Access Mode. The first such open request will succeed and return with the device in an initialized state. Subsequent such open requests will also succeed, but will not alter the device state. Once opened in Shared Access Mode, device access remains in this mode until all Shared Access Mode open requests release the device with a corresponding close request.

Exclusive Access Mode:

If the `share` argument is zero the device is opened in Exclusive Access Mode. In this mode, only one application at a time can access the device. The first such open request will succeed and return with the device in an initialized state. Subsequent open requests, regardless of the `share` argument value, will fail until the device is released with a corresponding close request.

4.1.7. `sio4_hdlc_rx_flush()`

This function flushes the entire receive side of the channel, both hardware and software wise, in case the application receives status indicating that data has been lost or corrupted. This may be called for in cases of a data overrun, a frame overrun, a CRC error or any other condition as reported when the frame is read. All receive data is discarded when this call is made. This includes data in the library's buffer, data in the SIO4's Rx FIFO, and data in the USC receiver.

NOTE: While a device file descriptor can be used with both the library's read services and the driver's read services, it is recommended that an application not use both simultaneously.

NOTE: All Rx Frame and Rx Flush requests are serialized and processed on a first come first served basis.

Prototype

```
int sio4_hdlc_rx_flush(int fd);
```

Argument	Description
<code>fd</code>	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.8. `sio4_hdlc_rx_frame()`

This function requests that an HDLC frame be read from the serial channel. The request will return either when it has been fulfilled or the read timeout expires, whichever occurs first. This is a blocking call.

Always consult the referenced structure's fields for completion status. This structure will always indicate the number of bytes retrieved, even with a failure return status. The status flags are set by the HDLC Protocol Library and may represent post data transfer status.

NOTE: While a device file descriptor can be used with both the library's read services and the driver's read services, it is recommended that an application not use both simultaneously.

NOTE: All Rx Frame and Rx Flush requests are serialized and processed on a first come first served basis.

NOTE: On a device handle obtained for device index -1 the data will be returned as if it were an HDLC frame.

Prototype

```
int sio4_hdlc_rx_frame(int fd, sio4_hdlc_rx_frame_t* rx);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
rx	This structure provides information to the library and is where information is returned by the library. See below.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

Data Type

This structure is used when reading an HDLC frame from a serial channel. All fields must be initialized before passing this structure to the protocol library.

```
typedef struct
{
    // All fields are initialized to zero before the call.
    // These are all filled in by the library after the transfer.
    u32      status;      // See documentation.
    u32      size;        // byte count without CRC and RSB
    s32      last;        // SIO4_HDLC_RX_LAST_CHAR_LEN_*
    u32      received;    // byte count with CRC and RSB
    u16      rsb;         // Receive Status Block (RSB)
    // "size + CRC and RSB" should equal "received".
    // They may be different in case of an error condition.
    u8       buffer[0xFFFF + 2]; // data + CRC + 16-bit RSB
} sio4_hdlc_rx_frame_t;
```

Field	Description														
status	This field reports status information about the transfer. This is reported by the library with each frame read request. The application must initialize this to zero. Valid bitwise options are given in the following tables.														
size	This is the size of the frame, excluding the CRC and the RSB. The application must initialize this to zero. *														
last	<p>This is the size of the last character of the frame. The application must initialize this to zero. The length specified does <u>not</u> include the Parity Bit Parity is enabled. Valid values are given in the table below. †</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_RX_LAST_CHAR_LEN_1</td><td>It is 1-bit wide.</td></tr> <tr> <td>SIO4_HDLC_RX_LAST_CHAR_LEN_2</td><td>It is 2-bits wide.</td></tr> <tr> <td>SIO4_HDLC_RX_LAST_CHAR_LEN_3</td><td>It is 3-bits wide.</td></tr> <tr> <td>SIO4_HDLC_RX_LAST_CHAR_LEN_4</td><td>It is 4-bits wide.</td></tr> <tr> <td>SIO4_HDLC_RX_LAST_CHAR_LEN_5</td><td>It is 5-bits wide.</td></tr> <tr> <td>SIO4_HDLC_RX_LAST_CHAR_LEN_6</td><td>It is 6-bits wide.</td></tr> </table>	Value	Description	SIO4_HDLC_RX_LAST_CHAR_LEN_1	It is 1-bit wide.	SIO4_HDLC_RX_LAST_CHAR_LEN_2	It is 2-bits wide.	SIO4_HDLC_RX_LAST_CHAR_LEN_3	It is 3-bits wide.	SIO4_HDLC_RX_LAST_CHAR_LEN_4	It is 4-bits wide.	SIO4_HDLC_RX_LAST_CHAR_LEN_5	It is 5-bits wide.	SIO4_HDLC_RX_LAST_CHAR_LEN_6	It is 6-bits wide.
Value	Description														
SIO4_HDLC_RX_LAST_CHAR_LEN_1	It is 1-bit wide.														
SIO4_HDLC_RX_LAST_CHAR_LEN_2	It is 2-bits wide.														
SIO4_HDLC_RX_LAST_CHAR_LEN_3	It is 3-bits wide.														
SIO4_HDLC_RX_LAST_CHAR_LEN_4	It is 4-bits wide.														
SIO4_HDLC_RX_LAST_CHAR_LEN_5	It is 5-bits wide.														
SIO4_HDLC_RX_LAST_CHAR_LEN_6	It is 6-bits wide.														

	SIO4_HDLC_RX_LAST_CHAR_LEN_7	It is 7-bits wide.
	SIO4_HDLC_RX_LAST_CHAR_LEN_8	It is 8-bits wide.
received	This is the size of the frame, including the CRC and the RSB. The application must initialize this to zero. *	
rsb	This is the value reported by the USC receiver in the RSB. This is provided for informational purposes only. The application must initialize this to zero. *	
buffer	The frame content is placed in this buffer.	

* The CRC is appended to the frame by the transmitter. The RSB (Receive Status Block) is 16-bits of data appended to the frame by the USC receiver. This is used by the API to acquire status information of the received frame. The RSB is reported to the application for informational purposes only.

† The last field value excludes the Parity bit. However, this is uncommon as parity checking is not normally used with HDLC.

This table lists the bitwise flags used in the structure's `status` field.

NOTE: For additional information refer to function `_eval_rx_status()` in the `hdlcc2c` sample application's `transfer.c` file.

This following set of flags represent status computed by the HDLC Protocol Library based on status generated in response to interrupts (see below) and status reported by the USC in the Receive Status Block (see further below).

Value	Description
SIO4_HDLC_RX_STAT_ERROR_BIG	The current frame's content exceeds the allowable frame size. The input stream is likely corrupt. *
SIO4_HDLC_RX_STAT_ERROR_PART	The current frame was only partially received. The input stream is likely corrupt. *
SIO4_HDLC_RX_STAT_FRAME_FULL	The current frame was received in its entirety. <i>This is the primary status to look for to verify that a frame has been received.</i>
SIO4_HDLC_RX_STAT_OVER	The SIO4 channel's external Rx FIFO has experienced an overrun. Data has been lost. *
SIO4_HDLC_RX_STAT_TIMEOUT	A complete frame was not received within the Rx I/O Timeout period. If the receiver began to receive a frame, then that frame is incomplete. If so, then the input stream is now likely corrupt. *
SIO4_HDLC_RX_STAT_UNDER	The SIO4 channel's external Rx FIFO was read when it was empty. The input stream may be corrupt. *

This following set of flags represent status gathered in response to interrupts generated by the USC. The receive frame logic uses some of these to guide reception of the frame.

Value	Description
SIO4_HDLC_RX_STAT_F_ABORT	An interrupt was generated because an abort sequence was detected on the Cable Rx Data line. The data stream may be corrupt. *
SIO4_HDLC_RX_STAT_F_ABORT_PE	An interrupt was generated because an abort sequence was detected on the Cable Rx Data line. The data stream may be corrupt. * †
SIO4_HDLC_RX_STAT_F_DPLL_DESYN	An interrupt was generated because the DPLL became desynchronized. Data may have been lost. *
SIO4_HDLC_RX_STAT_F_RCC_UNDER	An interrupt was generated because software read from the USC's RCC FIFO when it was empty. The input stream may be corrupt. *

SIO4_HDLC_RX_STAT_F_RX_BOUND	An interrupt was generated for the last frame character preceding an Abort sequence or an ending Flag. This is normal operation and is not indicative of an error.
SIO4_HDLC_RX_STAT_F_RX_FRAME	An interrupt was generated when the receiver detected the end of a frame. This is normal operation and is not indicative of an error.
SIO4_HDLC_RX_STAT_F_RX_OVERRUN	An interrupt was generated because the internal USC Rx data FIFO experienced an overrun. Data has been lost. *
SIO4_HDLC_RX_STAT_F_SW_OVERRUN	The driver's internal software RCC FIFO experienced an overrun. The input stream has become corrupt following the current frame. *†‡

This following set of flags represent status reported by the USC in the 16-bit Receive Status Block used by the HPDC Protocol Library.

Value	Description
SIO4_HDLC_RX_STAT_R_CRC_FE	The USC detected a CRC error. The input stream is corrupt. *
SIO4_HDLC_RX_STAT_R_EOF	The USC detected the end of a frame. This is normal operation and is not indicative of an error.
SIO4_HDLC_RX_STAT_R_FIFO_OVER	The USC internal Rx data FIFO experiencing an overrun. Data has been lost. *
SIO4_HDLC_RX_STAT_R_FRAME_SHORT	The USC detected premature termination of a frame. The input stream is corrupt. *
SIO4_HDLC_RX_STAT_R_PE	The USC detected a parity error. The input stream is corrupt. * †
SIO4_HDLC_RX_STAT_R_RCCF_OVER	The USC RCC FIFO experiencing an overrun. Data has been lost. *

* Each of these status flags represents a condition that calls for resynchronization of the data stream. The current frame may be affected and possibly a number of subsequent frames.

† Each of these refers to a Parity Error having been detected. However, this is uncommon as parity checking is not normally used with HDLC.

‡ The generally indicates that a high number of very small frames is being received. The driver's internal RCC FIFO accommodates 256 frames.

4.1.9. sio4_hdlc_set()

This function configures an SIO4 channel according to the settings of the referenced `sio4_hdlc_t` structure. All fields are validated before any settings are applied.

NOTE: Before calling this function, the structure should be initialized by calling the `sio4_hdlc_init_data()` function (section 4.1.3, page 12).

Prototype

```
int sio4_hdlc_set(
    int                fd,
    const sio4_hdlc_t* hdlc,
    const char**       err);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
hdlc	This is the structure containing the settings to be applied (section 4.2.1, page 26).
err	In the event of an error this will be set to identify the source of the error. The caller may provide a NULL pointer.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.10. `sio4_hdlc_show()`

This function displays the content of the referenced `sio4_hdlc_t` structure to the screen. This is provided to assist debugging efforts.

Prototype

```
int sio4_hdlc_show(
    int          fd,
    const sio4_hdlc_t* hdlc,
    FILE*        file,
    const char**  err);
```

Argument	Description
<code>fd</code>	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
<code>hdlc</code>	This is the structure whose content will be displayed. (Refer to section 4.2.1, page 26.)
<code>file</code>	This is a file pointer to which the output is sent. If this is NULL, then no output is generated. If this is <code>stdout</code> , then the output is sent to the terminal window. Output will otherwise be written to the referenced file.
<code>err</code>	In the event of an error this will be set to identify the source of the error. The caller may provide a NULL pointer.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.11. `sio4_hdlc_tx_abort()`

This function immediately initiates a Tx Abort sequence on the cable's output data signal. The call does not wait for completion. The Abort sequence generated is the long version if the transmitter is configured to send the long abort sequence when it runs out of data in the middle of a frame. With any other configuration setting, the short abort sequence is sent.

NOTE: All Tx Abort and Tx Break requests are serialized and performed on a first-come, first-served basis. If Tx Abort or Tx Break requests are initiated too close to one another, then the USC may not fulfill requests which are effectively overlapping.

NOTE: If a Tx Abort request is made too close to the end of a frame, then the USC may end the frame prematurely. When this occurs, the USC may omit some frame data, the CRC Sent notification and the EOF Sent notification.

Prototype

```
int sio4_hdlc_tx_abort(int fd);
```

Argument	Description
<code>fd</code>	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.12. `sio4_hdlc_tx_break()`

This function immediately initiates a Break condition on the cable's output data signal of a caller specified duration. The call waits for completion before returning.

NOTE: All Tx Abort and Tx Break requests are serialized and performed on a first-come, first-served basis. If Tx Abort or Tx Break requests are initiated too close to one another, then the USC may not fulfill requests which are effectively overlapping.

NOTE: A Tx Break supersedes the Tx Data signal's normal operation, thus preventing the transmitter's normal activity from appearing at the cable interface. The transmitter continues to generate requested Tx Abort sequences and it continues to serialize available data from the Tx FIFO. But while the Tx Break is active the Abort and serialized data do not appear at the cable interface.

Prototype

```
int sio4_hdlc_tx_break(int fd, int ms);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
ms	This is the desired duration of the Tx Break. The valid range is from zero to 60,000 milliseconds.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.13. `sio4_hdlc_tx_flush()`

This function flushes the entire transmit side of the channel. This is typically done as part of an error recovery process in the event of a Tx frame error. All buffered transmit data is discarded when this call is made and all transmit status is cleared from the transmitter.

NOTE: All Tx Frame, Tx Flush and Tx Status requests are serialized and processed on a first come first served basis.

Prototype

```
int sio4_hdlc_tx_flush(int fd);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.14. sio4_hdlc_tx_frame()

This function requests that an HDLC frame be written to the serial channel. The request will return either when it has been fulfilled or the write timeout expires, whichever occurs first. This is a blocking call.

In all cases, applications must examine the `status` and the `sent` fields (see below) for a full picture of the completion status.

NOTE: The Tx FIFO Overrun condition, as an error, is checked prior to writing any data to the Tx FIFO.

NOTE: All Tx Frame, Tx Flush and Tx Status requests are serialized and processed on a first come first served basis.

NOTE: Successful completion of a Tx frame means all data was written to the Tx FIFO and any required waiting were all completed within the write I/O timeout period.

NOTE: The driver may poll when waiting for various status conditions. The wait interval is one system timer tick between checks.

Prototype

```
int sio4_hdlc_tx_frame(int fd, sio4_hdlc_tx_frame_t* tx);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
tx	This structure provides information to the library and is where information is returned by the library. See below.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

Data Type

This structure is used when writing frames to the serial channel. All fields must be initialized before passing this structure to the library.

```
typedef struct
{
    // These are filled in by the caller before the transfer.
    s32      preload; // Preload frame data in Tx FIFO?
    s32      send;    // number of bytes to send
    const void* src;   // Address, Control, and data
    s32      last;     // SIO4_HDLC_TX_LAST_CHAR_LEN *
    s32      wait;     // Wait on these status conditions.

    // These are filled in by the library after the transfer.
    s32      sent;     // Number of bytes transferred.
    u32      status;   // status conditions at end of service
} sio4_hdlc_tx_frame_t;
```

Field	Description																		
preload	If set to one, the library waits for the previous Tx Frame to complete then preloads the present frame data into the Tx FIFO before beginning transmission. If set to zero, the software immediately begins writing the frame's data to the Tx FIFO. For additional information refer to the Operations section (section 5.8, page 54).																		
send	This is the number of bytes to transfer from the <code>src</code> field. This refers to the number of total bytes (address, control and data) from the source buffer irrespective of the configured word size. The value must be from two to 0xFFFF. The specified size must accommodate the CRC being added, if enabled, without the frame size exceeding 0xFFFF bytes.																		
src	This is the source for the data to include in the frame. This cannot be NULL.																		
last	<p>This is the size of the last character of the frame. The application must specify this before requesting the transfer. Valid values are given in the table below. The length specified does <u>not</u> include the Parity Bit, if Parity is enabled. The active data bits for the last byte must be right justified.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_1</td><td>It is 1-bit wide.</td></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_2</td><td>It is 2-bits wide.</td></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_3</td><td>It is 3-bits wide.</td></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_4</td><td>It is 4-bits wide.</td></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_5</td><td>It is 5-bits wide.</td></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_6</td><td>It is 6-bits wide.</td></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_7</td><td>It is 7-bits wide.</td></tr> <tr> <td>SIO4_HDLC_TX_LAST_CHAR_LEN_8</td><td>It is 8-bits wide.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_LAST_CHAR_LEN_1	It is 1-bit wide.	SIO4_HDLC_TX_LAST_CHAR_LEN_2	It is 2-bits wide.	SIO4_HDLC_TX_LAST_CHAR_LEN_3	It is 3-bits wide.	SIO4_HDLC_TX_LAST_CHAR_LEN_4	It is 4-bits wide.	SIO4_HDLC_TX_LAST_CHAR_LEN_5	It is 5-bits wide.	SIO4_HDLC_TX_LAST_CHAR_LEN_6	It is 6-bits wide.	SIO4_HDLC_TX_LAST_CHAR_LEN_7	It is 7-bits wide.	SIO4_HDLC_TX_LAST_CHAR_LEN_8	It is 8-bits wide.
Value	Description																		
SIO4_HDLC_TX_LAST_CHAR_LEN_1	It is 1-bit wide.																		
SIO4_HDLC_TX_LAST_CHAR_LEN_2	It is 2-bits wide.																		
SIO4_HDLC_TX_LAST_CHAR_LEN_3	It is 3-bits wide.																		
SIO4_HDLC_TX_LAST_CHAR_LEN_4	It is 4-bits wide.																		
SIO4_HDLC_TX_LAST_CHAR_LEN_5	It is 5-bits wide.																		
SIO4_HDLC_TX_LAST_CHAR_LEN_6	It is 6-bits wide.																		
SIO4_HDLC_TX_LAST_CHAR_LEN_7	It is 7-bits wide.																		
SIO4_HDLC_TX_LAST_CHAR_LEN_8	It is 8-bits wide.																		
wait	<p>This is a bit field which tells the API Library what conditions to wait for before returning to the caller. The value may be zero. The status for the corresponding conditions is checked, but not cleared. See the <i>status</i> field below for additional descriptions. The field is cleared upon return except for the condition detected, if any. This field essentially reflects different levels of certainty that an application may require with respect to transmission of the frame.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td>This means do not wait for any completion status. This status indicates that all Tx frame data has been written to the device's main Tx FIFO.</td></tr> <tr> <td>SIO4_HDLC_TX_FRAME_STATUS_M_FIFO</td><td>Wait for the SIO4's main Tx FIFO to become empty. This status indicates that all Tx frame data has been handed over to the USC.</td></tr> <tr> <td>SIO4_HDLC_TX_FRAME_STATUS_U_FIFO</td><td>Wait for the USC's internal Tx FIFO to become empty. This status indicates that the last Tx frame byte is being processed by the USC.</td></tr> <tr> <td>SIO4_HDLC_TX_FRAME_STATUS_CRC *</td><td>Wait for the CRC Sent status to be asserted. This status indicates that the transmitter has completed sending of the CRC out the cable interface.</td></tr> <tr> <td>SIO4_HDLC_TX_FRAME_STATUS_EOF *</td><td>Wait for the EOF Sent status to be asserted. This status indicates that the transmitter has completed sending of the frame out the cable interface.</td></tr> </table> <p>* This status is produced for each frame segment should a Tx frame request not be transmitted as a single contiguous frame.</p>	Value	Description	0	This means do not wait for any completion status. This status indicates that all Tx frame data has been written to the device's main Tx FIFO.	SIO4_HDLC_TX_FRAME_STATUS_M_FIFO	Wait for the SIO4's main Tx FIFO to become empty. This status indicates that all Tx frame data has been handed over to the USC.	SIO4_HDLC_TX_FRAME_STATUS_U_FIFO	Wait for the USC's internal Tx FIFO to become empty. This status indicates that the last Tx frame byte is being processed by the USC.	SIO4_HDLC_TX_FRAME_STATUS_CRC *	Wait for the CRC Sent status to be asserted. This status indicates that the transmitter has completed sending of the CRC out the cable interface.	SIO4_HDLC_TX_FRAME_STATUS_EOF *	Wait for the EOF Sent status to be asserted. This status indicates that the transmitter has completed sending of the frame out the cable interface.						
Value	Description																		
0	This means do not wait for any completion status. This status indicates that all Tx frame data has been written to the device's main Tx FIFO.																		
SIO4_HDLC_TX_FRAME_STATUS_M_FIFO	Wait for the SIO4's main Tx FIFO to become empty. This status indicates that all Tx frame data has been handed over to the USC.																		
SIO4_HDLC_TX_FRAME_STATUS_U_FIFO	Wait for the USC's internal Tx FIFO to become empty. This status indicates that the last Tx frame byte is being processed by the USC.																		
SIO4_HDLC_TX_FRAME_STATUS_CRC *	Wait for the CRC Sent status to be asserted. This status indicates that the transmitter has completed sending of the CRC out the cable interface.																		
SIO4_HDLC_TX_FRAME_STATUS_EOF *	Wait for the EOF Sent status to be asserted. This status indicates that the transmitter has completed sending of the frame out the cable interface.																		

sent	This is the number of bytes from the <code>src</code> field that have been transferred to the SIO4. This must be initialized to zero.	
status	<p>This is a bit field which reports status information about the transfer. Valid bit values are given in the table below. This must be initialized to zero.</p> <p>NOTE: These Tx Status macros are defined in <code>sio4_usc.h</code>.</p> <p>NOTE: For additional information refer to function <code>_eval_tx_status()</code> in the <code>hdlcc2c</code> sample application's <code>transfer.c</code> file.</p>	
	Value	Description
	<code>SIO4_HDLC_TX_FRAME_STATUS_ABORT</code>	The Abort Sent status was asserted, meaning an Abort sequence was sent out the cable interface. No user action is required upon receiving this status.
	<code>SIO4_HDLC_TX_FRAME_STATUS_CRC</code>	The CRC Sent status was asserted, meaning the frame CRC was sent out the cable interface. No user action is required upon receiving this status.
	<code>SIO4_HDLC_TX_FRAME_STATUS_EOF</code>	The EOF Sent status was asserted, meaning a frame's closing Flag was sent out the cable interface. No user action is required upon receiving this status.
	<code>SIO4_HDLC_TX_FRAME_STATUS_M_FIFO</code>	The SIO4's main Tx FIFO is empty. No user action is required upon receiving this status.
	<code>SIO4_HDLC_TX_FRAME_STATUS_M_OVER</code>	There was a main Tx FIFO overrun. The status is reported, but not cleared. This is an error condition and requires some type recovery action. Applications should never see this status.
	<code>SIO4_HDLC_TX_FRAME_STATUS_PREAMB</code>	The Preamble Sent status was asserted, meaning a frame preamble was sent out the cable interface. No user action is required upon receiving this status.
	<code>SIO4_HDLC_TX_FRAME_STATUS_SHORT</code>	A Tx Frame ended prematurely. This is because the Tx FIFO ran out of data when the USC required more data. This is an error condition and requires some type recovery action. For additional information refer to the Operations section (section 5.8, page 54).
	<code>SIO4_HDLC_TX_FRAME_STATUS_U_FIFO</code>	The USC's internal Tx FIFO is empty. No user action is required upon receiving this status.

4.1.15. `sio4_hdlc_tx_status()`

This function's purpose is to report the current transmitter status. When the function returns all applicable bits are set. Transient states are cleared. Error conditions are not cleared and require some sort of recovery operation.

NOTE: All Tx Frame, Tx Flush and Tx Status requests are serialized and processed on a first come first served basis.

Prototype

```
int sio4_hdlc_tx_status(int fd, s32* status);
```

Argument	Description		
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).		
status	This argument is used by the library to report information about the operation and the state of the device at the end of the operation. The <code>status</code> argument will return with one flag set, representing the first condition.		
	Values	Passed to Driver	Returned By Driver
	-1	This is an error.	The service is unsupported or HDLC Tx frame processing is disabled.
	SIO4 HDLC TX FRAME STATUS ABORT	The Abort Sent status was asserted.	
	SIO4 HDLC TX FRAME STATUS CRC	The CRC Sent status was asserted.	
	SIO4 HDLC TX FRAME STATUS EOF	The EOF Sent status was asserted.	
	SIO4 HDLC TX FRAME STATUS M FIFO	The SIO4's main Tx FIFO is empty.	
	SIO4 HDLC TX FRAME STATUS M OVER	There was a main Tx FIFO overrun.	
	SIO4 HDLC TX FRAME STATUS PREAMB	The Preamble Sent status was asserted.	
	SIO4 HDLC TX FRAME STATUS SHORT	A Tx Frame ended prematurely.	
	SIO4 HDLC TX FRAME STATUS U FIFO	The USC's internal Tx FIFO is empty.	

The function return values are as follows.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.1.16. sio4_hdlc_tx_wait()

This function's purpose is to wait for the first of any of the status conditions specified by the flags given in the below table.

NOTE: The driver polls for the specified conditions and waits one system timer tick between checks.

Prototype

```
int sio4_hdlc_tx_wait(int fd, s32* status);
```

Argument	Description		
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).		
flags	This argument is used by the library to report information about the operation and the state of the device at the end of the operation. The <code>flags</code> argument will return with one or more of the following values set.		
	Values	Passed to Driver	Returned By Driver
	-1	This is an	The service is

		error.	unsupported or HDLC Tx frame processing is disabled.
	SIO4_HDLC_TX_FRAME_STATUS_ABORT	Wait for the Abort Sent status to be asserted.	
	SIO4_HDLC_TX_FRAME_STATUS_CRC	Wait for the CRC Sent status to be asserted.	
	SIO4_HDLC_TX_FRAME_STATUS_EOF	Wait for the EOF Sent status to be asserted.	
	SIO4_HDLC_TX_FRAME_STATUS_M_FIFO	Wait for the SIO4's main Tx FIFO to become empty.	
	SIO4_HDLC_TX_FRAME_STATUS_M_OVER	Wait for a main Tx FIFO overrun.	
	SIO4_HDLC_TX_FRAME_STATUS_PREAMB	Wait for the Preamble Sent status to be asserted.	
	SIO4_HDLC_TX_FRAME_STATUS_SHORT	Wait for the USC transmitter to report that a frame has been subdivided.	
	SIO4_HDLC_TX_FRAME_STATUS_U_FIFO	Wait for the USC's internal Tx FIFO to become empty.	

The function return values are as follows.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative <code>errno.h</code> value.

4.2. Data Structures

The library header file is `sio4_hdlc.h`. Including this header in a source file gives the source the full library and driver interface as it includes the driver header files `sio4.h` and `sio4_usc.h`. The library header defines the complete HDLC interface offered by the library. The interface includes several functions, a few structures, and numerous macros. The data structures and associated macros are described below.

4.2.1. `sio4_hdlc_t`

This structure contains all of the parameters used to configure an SIO4 channel for HDLC operation. The structure is initialized with default values by calling the `sio4_hdlc_init_data()` function (section 4.1.3, page 12). Following this call, applications make changes to this structure's content according to their own requirements. Afterwards, the structure is passed to the `sio4_hdlc_set()` function (section 4.1.9, page 19) where the settings are applied to the board.

```
typedef struct
{
    struct
    {
        s32    ref;
        s32    prog;
    } osc;

    struct
    {
        s32    enable;        // PSRCR D31
        s32    mode;          // PSRCR D28, DCE or DTE
        s32    protocol;      // PSRSR D24-D27
    }
}
```

```

s32      txc;          // PSRCR D6-D8
s32      txd;          // PSRCR D19-D20
s32      txaux;        // PSRCR D17-D18
s32      dcd;          // PSRCR D15-D16
s32      dtr_dsr;      // PSRCR D21-D22
s32      rts;          // PSRCR D13-D14

struct
{
    s32 mode;          // PSRCR D23, D29
} loopback;

struct
{
    s32 enable;        // PSRCR D30
} term;

struct
{
    s32 txc;           // CCR 0x3333
    s32 txd_cts;       // CSR D2-D3
    s32 rxc;           // CCR 0xCCCC
    s32 rxd_dcd;       // CSR D4-D5
} legacy;

} cable;

struct
{
    s32 mode;          // USC CMR D8-D11
    s32 enable;        // USC TMR D0-D1
    s32 char_len;      // USC TMR D2-D4
    s32 encoding;      // USC TMR D13-D15
    s32 bit_rate;      // reflects sio4_hdlc_init_t.tx_bit_rate
    s32 idle_cond;     // USC TCSR D8-D10
    s32 share_0;       // USC CMR D12
    s32 underrun;      // USC CMR D14-D15
    s32 wait_underrun; // USC TCSR D11

    struct
    {
        s32 enable;    // USC TMR D9
        s32 type;      // USC TMR D11-D12
        s32 preset;    // USC TMR D10
        s32 on_end;    // USC TMR D8
    } crc;

    struct
    {
        s32 enable;    // USC TMR D5
        s32 type;      // USC TMR D6-D7
    } parity;

    struct
    {
        s32 enable;    // USC CMR D13
        s32 flag;      // USC CCR D12
    }

```

```

        s32 pattern;      // USC CCR D8-D9
        s32 length;      // USC CCR D10-D11
    } preamble;

    struct
    {
        s32 size;          // FSR D0-D15, read-only
        s32 ae;            // TAR D0-D15
        s32 af;            // TAR D16-D31
        s32 empty_cfg;     // CSR D18, D26
        s32 space_cfg;     // CSR D4-D5, else Rx 2x
    } fifo;

    struct
    {
        s32 dma_thresh;    // See notes.
        s32 mode;
        s32 pio_thresh;
        s32 timeout;
        s32 overrun;
    } io;

} tx;

struct
{
    s32    mode;           // USC CMR D0-D3
    s32    adrs;           // USC RSR D0-D7
    s32    adrs_ctrl;      // USC CMR D4-D7
    s32    enable;         // USC RMR D0-D1
    s32    char_len;       // USC RMR D2-D4
    s32    encoding;       // USC RMR D13-D15
    s32    size_limit;     // USC RCLR
    s32    bit_rate;       // reflects sio4_hdlc_init_t.rx_bit_rate
    s32    queue_abort;    // USC RMR D8
    s32    sync_byte;      // SBR D0-D7
    s32    status_word;    // CSR D3

    struct
    {
        s32 enable;        // USC RMR D9
        s32 type;          // USC RMR D11-D12
        s32 preset;        // USC RMR D10
    } crc;

    struct
    {
        s32 enable;        // USC RMR D5
        s32 type;          // USC RMR D6-D7
    } parity;

    struct
    {
        s32 size;          // FSR D16-D21, read-only
        s32 ae;            // RAR D0-D15
        s32 af;            // RAR D16-D31
        s32 full_cfg;      // BCR D8
    }

```

```

    } fifo;

    struct
    {
        s32 dma_thresh; // See notes.
        s32 mode;
        s32 pio_thresh;
        s32 timeout;
        s32 overrun;
        s32 underrun;
    } io;

    struct
    {
        s32 enable; // CSR D2
        s32 clk_src; // BCR D22
    } time_stamp;

} rx;

struct
{
    s32 mode; // USC CCAR D8-D9
    s32 txd; // USC IOCR D6-D7
    s32 cts; // PSRCR D9-D10 + USC IOCR D14-D15
    s32 cts_legacy; // USC IOCR D14-D15
    s32 dcd; // PSRCR D11-D12 + USC IOCR D12-D13
    s32 dcd_legacy; // USC IOCR D12-D13

    // All of the follong USC fields are initialized
    // by sio4_hdlc_init_data() based on the content of the
    // sio4_hdlc_init_t structure.

    struct
    {
        s32 clk_src; // USC CMCR D3-D5
        s32 txc; // PSRCR D0-D2 + USC IOCR D3-D5
        s32 txc_legacy; // USC IOCR D3-D5
    } tx;

    struct
    {
        s32 clk_src; // USC CMCR D0-D2
        s32 rxc; // PSRCR D3-D5 + USC IOCR D0-D2
        s32 rxc_legacy; // USC IOCR D0-D2
    } rx;

    struct
    {
        s32 enable; // USC HCR D0
        s32 clk_src; // USC CMCR D8-D9
        s32 divider; // USC TC1R D0-D15
        s32 mode; // USC HCR D1
    } brg0;

    struct
    {

```

```

        s32 enable;        // USC HCR D4
        s32 clk_src;       // USC CMCR D10-D11
        s32 divider;      // USC TCOR D0-D15
        s32 mode;         // USC HCR D5
    } brg1;

    struct
    {
        s32 clk_src;       // USC CMCR D12-D13
        s32 rate;         // USC HCR D14-D15
    } ctr0;

    struct
    {
        s32 clk_src;       // USC CMCR D14-D15
        s32 rate_src;     // USC HCR D13 + ...
    } ctrl1;

    struct
    {
        s32 clk_src;       // USC CMCR D6-D7
        s32 mode;         // USC HCR D8-D9
        s32 rate;         // USC HCR D10-D11
        s32 edge;         // USC CCSR D8-D9
    } dpll;

    } usc;

} sio4_hdlc_t;

```

4.2.1.1. sio4_hdlc_t.osc

This section describes the structure's oscillator configuration fields.

Field	Description
osc	This structure configures the oscillator interface.
osc.ref	This field specifies the frequency of the fixed onboard reference oscillator. The default is 20MHz. However, as this parameter refers to a fixed resource on the board, the default of 20MHz is used only if there is a problem accessing the setting from the driver.
osc.prog	This field specifies the desired programmable oscillator frequency. This is essentially the clock frequency provided by the onboard programmable oscillator to the USC. The default is 20MHz.

4.2.1.2. sio4_hdlc_t.cable

This section describes the structure's cable configuration fields.

Field	Description	
cable	This structure configures the cable interface.	
cable. enable	This field either enables or disables the cable transceivers. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_CABLE_ENABLE_NO	Leave the cable transceivers disabled.
	SIO4_HDLC_CABLE_ENABLE_YES	Enable the cable transceiver. This is the default. This option disables all legacy cable related settings.

cable. mode	This field specifies the arrangement of the signals on the cable interface. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_CABLE_MODE_DCE	Select the DCE cable signal configuration.
	SIO4_HDLC_CABLE_MODE_DTE	Select the DTE cable signal configuration. This is the default.
cable. protocol	This field specifies the cable transceiver configuration. The options available depend on the board's transceiver capabilities. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_CABLE_PROTOCOL_RS232	This selects the RS232 protocol. This is the default.
	SIO4_HDLC_CABLE_PROTOCOL_RS422_423_1	This selects the RS422/RS423 mixed protocol version 1.
	SIO4_HDLC_CABLE_PROTOCOL_RS422_423_2	This selects the RS422/RS423 mixed protocol version 2.
	SIO4_HDLC_CABLE_PROTOCOL_RS422_RS485	This selects the RS422/RS485 mixed protocol.
	SIO4_HDLC_CABLE_PROTOCOL_RS423	This selects the RS423 protocol.
	SIO4_HDLC_CABLE_PROTOCOL_RS530	This selects the RS530 protocol, version 1.
	SIO4_HDLC_CABLE_PROTOCOL_RS530A	This selects the RS530 protocol, version 2.
	SIO4_HDLC_CABLE_PROTOCOL_V35	This selects the V.35 protocol, version 1.
	SIO4_HDLC_CABLE_PROTOCOL_V35A	This selects the V.35 protocol, version 2.
cable. txc	This field specifies the configuration of the cable's Tx Clock signal. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_CABLE_TXC_OUT_0	This drives the signal low.
	SIO4_HDLC_CABLE_TXC_OUT_1	This drives the signal high.
	SIO4_HDLC_CABLE_TXC_OUT_CBL_RXA	This drives the signal from what appears at the cable's Rx Aux signal.
	SIO4_HDLC_CABLE_TXC_OUT_CBL_RXC	This drives the signal from what appears at the cable's Rx Clock signal.
	SIO4_HDLC_CABLE_TXC_OUT_OSC	This drives the signal from the onboard oscillator.
	SIO4_HDLC_CABLE_TXC_OUT_OSC_INV	This drives the signal from the inverted form of the onboard oscillator.
	SIO4_HDLC_CABLE_TXC_OUT_USC_RXC	This drives the signal from what appears at the USC's Rx Clock pin.
	SIO4_HDLC_CABLE_TXC_OUT_USC_TXC	This drives the signal from what appears at the USC's Tx Clock pin. This is the default.
cable. txd	This field specifies the configuration of the cable's Tx Data signal. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_CABLE_TXD_OUT_0	This drives the signal low.
	SIO4_HDLC_CABLE_TXD_OUT_1	This drives the signal high.
	SIO4_HDLC_CABLE_TXD_OUT_USC_TXD	This drives the signal from what appears at the USC's Tx Data pin. This is the default.
cable.	This field specifies the configuration of the cable's Tx Aux signal. Valid values are given in the	

txaux	<p>table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_CABLE_TXAUX_OUT_0</td><td>This drives the signal low.</td></tr> <tr> <td>SIO4_HDLC_CABLE_TXAUX_OUT_1</td><td>This drives the signal high.</td></tr> <tr> <td>SIO4_HDLC_CABLE_TXAUX_OUT_OSC</td><td>This drives the signal from the onboard oscillator.</td></tr> <tr> <td>SIO4_HDLC_CABLE_TXAUX_TRI</td><td>This tri-states the drive segment of the transceivers. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_CABLE_TXAUX_OUT_0	This drives the signal low.	SIO4_HDLC_CABLE_TXAUX_OUT_1	This drives the signal high.	SIO4_HDLC_CABLE_TXAUX_OUT_OSC	This drives the signal from the onboard oscillator.	SIO4_HDLC_CABLE_TXAUX_TRI	This tri-states the drive segment of the transceivers. This is the default.
Value	Description										
SIO4_HDLC_CABLE_TXAUX_OUT_0	This drives the signal low.										
SIO4_HDLC_CABLE_TXAUX_OUT_1	This drives the signal high.										
SIO4_HDLC_CABLE_TXAUX_OUT_OSC	This drives the signal from the onboard oscillator.										
SIO4_HDLC_CABLE_TXAUX_TRI	This tri-states the drive segment of the transceivers. This is the default.										
cable.dcd	<p>This field specifies the cable DCD signal source when the cable signal is driven. Valid values are given in the table below.</p> <p>NOTE: Refer to the <code>usc.dcd</code> field (section 4.2.1.5, page 42) for affecting the cable signal's driven state.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_CABLE_DCD_OUT_0</td><td>This drives the signal low.</td></tr> <tr> <td>SIO4_HDLC_CABLE_DCD_OUT_1</td><td>This drives the signal high.</td></tr> <tr> <td>SIO4_HDLC_CABLE_DCD_OUT_RTS</td><td>This drives the signal from the Rx FIFO Almost Full status.</td></tr> <tr> <td>SIO4_HDLC_CABLE_DCD_OUT_USC_DCD</td><td>This drives the signal from what appears at the USC's DCD pin. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_CABLE_DCD_OUT_0	This drives the signal low.	SIO4_HDLC_CABLE_DCD_OUT_1	This drives the signal high.	SIO4_HDLC_CABLE_DCD_OUT_RTS	This drives the signal from the Rx FIFO Almost Full status.	SIO4_HDLC_CABLE_DCD_OUT_USC_DCD	This drives the signal from what appears at the USC's DCD pin. This is the default.
Value	Description										
SIO4_HDLC_CABLE_DCD_OUT_0	This drives the signal low.										
SIO4_HDLC_CABLE_DCD_OUT_1	This drives the signal high.										
SIO4_HDLC_CABLE_DCD_OUT_RTS	This drives the signal from the Rx FIFO Almost Full status.										
SIO4_HDLC_CABLE_DCD_OUT_USC_DCD	This drives the signal from what appears at the USC's DCD pin. This is the default.										
cable.dtr_dsr	<p>This field specifies the configuration of the cable's DTR/DSR signal. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_CABLE_DTR_DSR_OUT_0</td><td>This drives the signal low.</td></tr> <tr> <td>SIO4_HDLC_CABLE_DTR_DSR_OUT_1</td><td>This drives the signal high.</td></tr> <tr> <td>SIO4_HDLC_CABLE_DTR_DSR_IN</td><td>This configures the signal as an input.</td></tr> <tr> <td>SIO4_HDLC_CABLE_DTR_DSR_TRI</td><td>This tri-states the drive segment of the transceivers. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_CABLE_DTR_DSR_OUT_0	This drives the signal low.	SIO4_HDLC_CABLE_DTR_DSR_OUT_1	This drives the signal high.	SIO4_HDLC_CABLE_DTR_DSR_IN	This configures the signal as an input.	SIO4_HDLC_CABLE_DTR_DSR_TRI	This tri-states the drive segment of the transceivers. This is the default.
Value	Description										
SIO4_HDLC_CABLE_DTR_DSR_OUT_0	This drives the signal low.										
SIO4_HDLC_CABLE_DTR_DSR_OUT_1	This drives the signal high.										
SIO4_HDLC_CABLE_DTR_DSR_IN	This configures the signal as an input.										
SIO4_HDLC_CABLE_DTR_DSR_TRI	This tri-states the drive segment of the transceivers. This is the default.										
cable.rts	<p>This field specifies the configuration of the cable's RTS signal. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_CABLE_RTS_OUT_0</td><td>This drives the signal low.</td></tr> <tr> <td>SIO4_HDLC_CABLE_RTS_OUT_1</td><td>This drives the signal high.</td></tr> <tr> <td>SIO4_HDLC_CABLE_RTS_OUT_CTS</td><td>This drives the signal from what appears at the USC's RTS pin.</td></tr> <tr> <td>SIO4_HDLC_CABLE_RTS_OUT_RTS</td><td>This drives the signal from the Rx FIFO Almost Full status. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_CABLE_RTS_OUT_0	This drives the signal low.	SIO4_HDLC_CABLE_RTS_OUT_1	This drives the signal high.	SIO4_HDLC_CABLE_RTS_OUT_CTS	This drives the signal from what appears at the USC's RTS pin.	SIO4_HDLC_CABLE_RTS_OUT_RTS	This drives the signal from the Rx FIFO Almost Full status. This is the default.
Value	Description										
SIO4_HDLC_CABLE_RTS_OUT_0	This drives the signal low.										
SIO4_HDLC_CABLE_RTS_OUT_1	This drives the signal high.										
SIO4_HDLC_CABLE_RTS_OUT_CTS	This drives the signal from what appears at the USC's RTS pin.										
SIO4_HDLC_CABLE_RTS_OUT_RTS	This drives the signal from the Rx FIFO Almost Full status. This is the default.										
cable.loopback	This structure configures the cable's loopback feature.										
cable.loopback.mode	<p>This field specifies the loopback mode. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_LOOPBACK_MODE_DISABLE</td><td>This disables loopback operation. This is the default.</td></tr> <tr> <td>SIO4_HDLC_LOOPBACK_MODE_EXTERNAL</td><td>This selects the external loopback mode.*†</td></tr> <tr> <td>SIO4_HDLC_LOOPBACK_MODE_INTERNAL</td><td>This selects the internal loopback mode.†</td></tr> </table> <p>* If external loopback mode is requested but not available, then the internal loopback mode is selected.</p> <p>† Both loopback modes are performed onboard the SIO4; internal inside the USC, external at the cable interface. For external mode, enable the transceivers and disconnect cabling.</p>	Value	Description	SIO4_HDLC_LOOPBACK_MODE_DISABLE	This disables loopback operation. This is the default.	SIO4_HDLC_LOOPBACK_MODE_EXTERNAL	This selects the external loopback mode.*†	SIO4_HDLC_LOOPBACK_MODE_INTERNAL	This selects the internal loopback mode.†		
Value	Description										
SIO4_HDLC_LOOPBACK_MODE_DISABLE	This disables loopback operation. This is the default.										
SIO4_HDLC_LOOPBACK_MODE_EXTERNAL	This selects the external loopback mode.*†										
SIO4_HDLC_LOOPBACK_MODE_INTERNAL	This selects the internal loopback mode.†										
cable.term	This structure configures the cable's termination feature. The operation of this feature depends on the selected cable protocol.										

cable. term. enable	This field specifies the configuration of the transceiver's built-in termination capabilities. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_CABLE_TERM_ENABLE_NO	The built-in termination is disabled. This is the default.
	SIO4_HDLC_CABLE_TERM_ENABLE_YES	The built-in termination is enabled.
cable. legacy	This structure configures the cable's legacy interface feature. These fields are utilized if the board DCE/DTE cable configuration feature is absent or unused. Please also read Cable Configuration Modes (section 5.10, page 61).	
cable. legacy. txc	This field specifies the legacy configuration of the cable's Tx Clock signal. Valid values are given in the table below. Please also read Cable Configuration Modes (section 5.10, page 61).	
	Value	Description
	SIO4_HDLC_CABLE_LEGACY_TXC_DISABLE	This disables the Tx Clock signal.
	SIO4_HDLC_CABLE_LEGACY_TXC_BOTH	This drives the Tx Clock signal on both the upper and lower group of pins.
	SIO4_HDLC_CABLE_LEGACY_TXC_LOW	This drives the Tx Clock signal on the lower group of pins.
	SIO4_HDLC_CABLE_LEGACY_TXC_UP	This drives the Tx Clock signal on the upper group of pins. This is the default.
cable. legacy. txd_cts	This field specifies the legacy configuration of the cable's Tx Data and CTS signals. Valid values are given in the table below. Please also read Cable Configuration Modes (section 5.10, page 61).	
	Value	Description
	SIO4_HDLC_CABLE_LEGACY_TXD_CTS_BOTH	This drives the signals on both the upper and lower group of pins.
	SIO4_HDLC_CABLE_LEGACY_TXD_CTS_LOW	This drives the signals on the lower group of pins.
	SIO4_HDLC_CABLE_LEGACY_TXD_CTS_TRI	This tri-states the signals.
	SIO4_HDLC_CABLE_LEGACY_TXD_CTS_UP	This drives the signals on the upper group of pins. This is the default.
cable. legacy. rxc	This field specifies the legacy configuration of the cable's Rx Clock signal. Valid values are given in the table below. Please also read Cable Configuration Modes (section 5.10, page 61).	
	Value	Description
	SIO4_HDLC_CABLE_LEGACY_RXC_DISABLE	This disables the Tx Clock signal.
	SIO4_HDLC_CABLE_LEGACY_RXC_LOW	This drives the Tx Clock signal on both the upper and lower group of pins. This is the default.
	SIO4_HDLC_CABLE_LEGACY_RXC_UP	This drives the Tx Clock signal on the lower group of pins.
cable. legacy. rxd_dcd	This field specifies the legacy configuration of the cable's Rx Data and DCD signals. Valid values are given in the table below. Please also read Cable Configuration Modes (section 5.10, page 61).	
	Value	Description
	SIO4_HDLC_CABLE_LEGACY_RXD_DCD_DISABLE	This disables the signals.
	SIO4_HDLC_CABLE_LEGACY_RXD_DCD_LOW	This uses the signals as inputs from the lower group of pins. This is the default.
	SIO4_HDLC_CABLE_LEGACY_RXD_DCD_UP	This uses the signals as inputs from the upper group of pins.

4.2.1.3. sio4_hdlc_t.tx

This section describes the structure's transmitter configuration fields.

Field	Description																		
tx	This structure configures the transmitter portion of the channel.																		
tx. mode	<p>This field specifies the transmitter's operating mode. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_MODE_HDLC</td><td>This selects the HDLC operating mode. This is the default and the only valid option for this library.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_MODE_HDLC	This selects the HDLC operating mode. This is the default and the only valid option for this library.														
Value	Description																		
SIO4_HDLC_TX_MODE_HDLC	This selects the HDLC operating mode. This is the default and the only valid option for this library.																		
tx. enable	<p>This field specifies if the transmitter is to be enabled. When configuration is begun (see <code>sio4_hdlc_set()</code>, section 4.1.9, page 19) the transmitter is initialized and disabled. The option in this field is applied towards the end of the configuration process. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_ENABLE_NO_AFTER</td><td>This disables the transmitter after it has finished the transmission in progress.</td></tr> <tr> <td>SIO4_HDLC_TX_ENABLE_NO_NOW</td><td>This disables the transmitter immediately.</td></tr> <tr> <td>SIO4_HDLC_TX_ENABLE_YES_NOW</td><td>This enables the transmitter immediately. This is the default.</td></tr> <tr> <td>SIO4_HDLC_TX_ENABLE_YES_W_AE</td><td>This enables the transmitter according to the state of any hardware flow control lines.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_ENABLE_NO_AFTER	This disables the transmitter after it has finished the transmission in progress.	SIO4_HDLC_TX_ENABLE_NO_NOW	This disables the transmitter immediately.	SIO4_HDLC_TX_ENABLE_YES_NOW	This enables the transmitter immediately. This is the default.	SIO4_HDLC_TX_ENABLE_YES_W_AE	This enables the transmitter according to the state of any hardware flow control lines.								
Value	Description																		
SIO4_HDLC_TX_ENABLE_NO_AFTER	This disables the transmitter after it has finished the transmission in progress.																		
SIO4_HDLC_TX_ENABLE_NO_NOW	This disables the transmitter immediately.																		
SIO4_HDLC_TX_ENABLE_YES_NOW	This enables the transmitter immediately. This is the default.																		
SIO4_HDLC_TX_ENABLE_YES_W_AE	This enables the transmitter according to the state of any hardware flow control lines.																		
tx. char_len	<p>This field specifies if the size of transmitted characters. The length specified <u>includes</u> the Parity Bit, if Parity is enabled. The data bits are the lower significant bits of the byte. (See the Z16C30 data book for exceptions.) Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_1</td><td>Characters are 1-bit in length.</td></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_2</td><td>Characters are 2-bits in length.</td></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_3</td><td>Characters are 3-bits in length.</td></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_4</td><td>Characters are 4-bits in length.</td></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_5</td><td>Characters are 5-bits in length.</td></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_6</td><td>Characters are 6-bits in length.</td></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_7</td><td>Characters are 7-bits in length.</td></tr> <tr> <td>SIO4_HDLC_TX_CHAR_LEN_8</td><td>Characters are 8-bits in length. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_CHAR_LEN_1	Characters are 1-bit in length.	SIO4_HDLC_TX_CHAR_LEN_2	Characters are 2-bits in length.	SIO4_HDLC_TX_CHAR_LEN_3	Characters are 3-bits in length.	SIO4_HDLC_TX_CHAR_LEN_4	Characters are 4-bits in length.	SIO4_HDLC_TX_CHAR_LEN_5	Characters are 5-bits in length.	SIO4_HDLC_TX_CHAR_LEN_6	Characters are 6-bits in length.	SIO4_HDLC_TX_CHAR_LEN_7	Characters are 7-bits in length.	SIO4_HDLC_TX_CHAR_LEN_8	Characters are 8-bits in length. This is the default.
Value	Description																		
SIO4_HDLC_TX_CHAR_LEN_1	Characters are 1-bit in length.																		
SIO4_HDLC_TX_CHAR_LEN_2	Characters are 2-bits in length.																		
SIO4_HDLC_TX_CHAR_LEN_3	Characters are 3-bits in length.																		
SIO4_HDLC_TX_CHAR_LEN_4	Characters are 4-bits in length.																		
SIO4_HDLC_TX_CHAR_LEN_5	Characters are 5-bits in length.																		
SIO4_HDLC_TX_CHAR_LEN_6	Characters are 6-bits in length.																		
SIO4_HDLC_TX_CHAR_LEN_7	Characters are 7-bits in length.																		
SIO4_HDLC_TX_CHAR_LEN_8	Characters are 8-bits in length. This is the default.																		
tx. encoding	<p>This field specifies if the encoding of the transmitted data. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_BI_MARK</td><td>This refers to Biphasic Mark encoding.</td></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_BI_LEVEL</td><td>This refers to Biphasic Level encoding.</td></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_BI_SPACE</td><td>This refers to Biphasic Space encoding.</td></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_D_BI_LEVEL</td><td>This refers to Differential Biphasic Level encoding.</td></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_NRZ</td><td>This refers to NRZ encoding.</td></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_NRZB</td><td>This refers to NRZB encoding.</td></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_NRZI_MARK</td><td>This refers to NRZI-Mark encoding.</td></tr> <tr> <td>SIO4_HDLC_TX_ENCODING_NRZI_SPACE</td><td>This refers to NRZI-Space encoding. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_ENCODING_BI_MARK	This refers to Biphasic Mark encoding.	SIO4_HDLC_TX_ENCODING_BI_LEVEL	This refers to Biphasic Level encoding.	SIO4_HDLC_TX_ENCODING_BI_SPACE	This refers to Biphasic Space encoding.	SIO4_HDLC_TX_ENCODING_D_BI_LEVEL	This refers to Differential Biphasic Level encoding.	SIO4_HDLC_TX_ENCODING_NRZ	This refers to NRZ encoding.	SIO4_HDLC_TX_ENCODING_NRZB	This refers to NRZB encoding.	SIO4_HDLC_TX_ENCODING_NRZI_MARK	This refers to NRZI-Mark encoding.	SIO4_HDLC_TX_ENCODING_NRZI_SPACE	This refers to NRZI-Space encoding. This is the default.
Value	Description																		
SIO4_HDLC_TX_ENCODING_BI_MARK	This refers to Biphasic Mark encoding.																		
SIO4_HDLC_TX_ENCODING_BI_LEVEL	This refers to Biphasic Level encoding.																		
SIO4_HDLC_TX_ENCODING_BI_SPACE	This refers to Biphasic Space encoding.																		
SIO4_HDLC_TX_ENCODING_D_BI_LEVEL	This refers to Differential Biphasic Level encoding.																		
SIO4_HDLC_TX_ENCODING_NRZ	This refers to NRZ encoding.																		
SIO4_HDLC_TX_ENCODING_NRZB	This refers to NRZB encoding.																		
SIO4_HDLC_TX_ENCODING_NRZI_MARK	This refers to NRZI-Mark encoding.																		
SIO4_HDLC_TX_ENCODING_NRZI_SPACE	This refers to NRZI-Space encoding. This is the default.																		
tx. bit_rate	This specifies the desired transmission bit rate. During the <code>sio4_hdlc_init_data()</code> call (section 4.1.3, page 12) this is computed from the <code>sio4_hdlc_init_t.tx_bit_rate</code> field provided to the call.																		

tx. idle_cond	<p>This field specifies what appears on the Tx Data cable signal while no data is being transmitted. Valid values are given in the table below.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>SIO4_HDLC_TX_IDLE_COND_0</td><td>The Tx Data signal outputs a continuous data “0” value.</td></tr> <tr> <td>SIO4_HDLC_TX_IDLE_COND_0_1</td><td>The Tx Data signal is alternately data “0” and data “1” values.</td></tr> <tr> <td>SIO4_HDLC_TX_IDLE_COND_1</td><td>The Tx Data signal outputs a continuous data “1” value.</td></tr> <tr> <td>SIO4_HDLC_TX_IDLE_COND_DEFAULT</td><td>The Tx Data signal is with the pattern that is the default for the selected serial protocol. This is the default.</td></tr> <tr> <td>SIO4_HDLC_TX_IDLE_COND_MARK</td><td>The Tx Data signal is held high.</td></tr> <tr> <td>SIO4_HDLC_TX_IDLE_COND_MARK_SPACE</td><td>The Tx Data signal is alternately driven with the high then low.</td></tr> <tr> <td>SIO4_HDLC_TX_IDLE_COND_SPACE</td><td>The Tx Data signal is held low.</td></tr> </tbody> </table>	Value	Description	SIO4_HDLC_TX_IDLE_COND_0	The Tx Data signal outputs a continuous data “0” value.	SIO4_HDLC_TX_IDLE_COND_0_1	The Tx Data signal is alternately data “0” and data “1” values.	SIO4_HDLC_TX_IDLE_COND_1	The Tx Data signal outputs a continuous data “1” value.	SIO4_HDLC_TX_IDLE_COND_DEFAULT	The Tx Data signal is with the pattern that is the default for the selected serial protocol. This is the default.	SIO4_HDLC_TX_IDLE_COND_MARK	The Tx Data signal is held high.	SIO4_HDLC_TX_IDLE_COND_MARK_SPACE	The Tx Data signal is alternately driven with the high then low.	SIO4_HDLC_TX_IDLE_COND_SPACE	The Tx Data signal is held low.
Value	Description																
SIO4_HDLC_TX_IDLE_COND_0	The Tx Data signal outputs a continuous data “0” value.																
SIO4_HDLC_TX_IDLE_COND_0_1	The Tx Data signal is alternately data “0” and data “1” values.																
SIO4_HDLC_TX_IDLE_COND_1	The Tx Data signal outputs a continuous data “1” value.																
SIO4_HDLC_TX_IDLE_COND_DEFAULT	The Tx Data signal is with the pattern that is the default for the selected serial protocol. This is the default.																
SIO4_HDLC_TX_IDLE_COND_MARK	The Tx Data signal is held high.																
SIO4_HDLC_TX_IDLE_COND_MARK_SPACE	The Tx Data signal is alternately driven with the high then low.																
SIO4_HDLC_TX_IDLE_COND_SPACE	The Tx Data signal is held low.																
tx. share_0	<p>This field specifies if the Flag patterns driven on the Tx Data signal during idle periods will share the intervening zero value. The transmitter never shares the zeros that appear at frame boundaries. Valid values are given in the table below.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>SIO4_HDLC_TX_SHARE_0_NO</td><td>Do not share the zero bit.</td></tr> <tr> <td>SIO4_HDLC_TX_SHARE_0_YES</td><td>Do share the zero bit. This is the default.</td></tr> </tbody> </table>	Value	Description	SIO4_HDLC_TX_SHARE_0_NO	Do not share the zero bit.	SIO4_HDLC_TX_SHARE_0_YES	Do share the zero bit. This is the default.										
Value	Description																
SIO4_HDLC_TX_SHARE_0_NO	Do not share the zero bit.																
SIO4_HDLC_TX_SHARE_0_YES	Do share the zero bit. This is the default.																
tx. underrun	<p>This field specifies what the transmitter will transmit when it needs data but none is present in its Tx FIFO. Valid values are given in the table below.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>SIO4_HDLC_TX_UNDERRUN_ABORT</td><td>The transmitter sends an Abort sequence.</td></tr> <tr> <td>SIO4_HDLC_TX_UNDERRUN_CRC_F</td><td>The transmitter sends the configured CRC followed by the Flag sequence. This is the default.</td></tr> <tr> <td>SIO4_HDLC_TX_UNDERRUN_EXT_A</td><td>The transmitter sends an Extended Abort sequence.</td></tr> <tr> <td>SIO4_HDLC_TX_UNDERRUN_FLAG</td><td>The transmitter sends the Flag sequence.</td></tr> </tbody> </table>	Value	Description	SIO4_HDLC_TX_UNDERRUN_ABORT	The transmitter sends an Abort sequence.	SIO4_HDLC_TX_UNDERRUN_CRC_F	The transmitter sends the configured CRC followed by the Flag sequence. This is the default.	SIO4_HDLC_TX_UNDERRUN_EXT_A	The transmitter sends an Extended Abort sequence.	SIO4_HDLC_TX_UNDERRUN_FLAG	The transmitter sends the Flag sequence.						
Value	Description																
SIO4_HDLC_TX_UNDERRUN_ABORT	The transmitter sends an Abort sequence.																
SIO4_HDLC_TX_UNDERRUN_CRC_F	The transmitter sends the configured CRC followed by the Flag sequence. This is the default.																
SIO4_HDLC_TX_UNDERRUN_EXT_A	The transmitter sends an Extended Abort sequence.																
SIO4_HDLC_TX_UNDERRUN_FLAG	The transmitter sends the Flag sequence.																
tx. wait_underrun	<p>This field specifies the transmitter’s response to a Tx Underrun condition.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>SIO4_HDLC_TX_WAIT_UNDERRUN_NO</td><td>The transmitter is to end the frame prematurely and resume transmission when possible. This is the default.</td></tr> <tr> <td>SIO4_HDLC_TX_WAIT_UNDERRUN_YES</td><td>The transmitter is to end the frame prematurely than wait before resuming transmission. If this option is selected, resuming data transmission is the application’s responsibility.</td></tr> </tbody> </table>	Value	Description	SIO4_HDLC_TX_WAIT_UNDERRUN_NO	The transmitter is to end the frame prematurely and resume transmission when possible. This is the default.	SIO4_HDLC_TX_WAIT_UNDERRUN_YES	The transmitter is to end the frame prematurely than wait before resuming transmission. If this option is selected, resuming data transmission is the application’s responsibility.										
Value	Description																
SIO4_HDLC_TX_WAIT_UNDERRUN_NO	The transmitter is to end the frame prematurely and resume transmission when possible. This is the default.																
SIO4_HDLC_TX_WAIT_UNDERRUN_YES	The transmitter is to end the frame prematurely than wait before resuming transmission. If this option is selected, resuming data transmission is the application’s responsibility.																
tx. crc	<p>This structure configures the transmitter’s use of a CRC at the end of a Frame.</p>																
tx. crc. enable	<p>This field enables or disables use of a CRC at the end of frames. Valid values are given in the table below.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>SIO4_HDLC_TX_CRC_ENABLE_NO</td><td>CRCs are <u>not</u> used.</td></tr> <tr> <td>SIO4_HDLC_TX_CRC_ENABLE_YES</td><td>CRCs are <u>used</u>. This is the default.</td></tr> </tbody> </table>	Value	Description	SIO4_HDLC_TX_CRC_ENABLE_NO	CRCs are <u>not</u> used.	SIO4_HDLC_TX_CRC_ENABLE_YES	CRCs are <u>used</u> . This is the default.										
Value	Description																
SIO4_HDLC_TX_CRC_ENABLE_NO	CRCs are <u>not</u> used.																
SIO4_HDLC_TX_CRC_ENABLE_YES	CRCs are <u>used</u> . This is the default.																

tx. crc. type	<p>This field selects the type of CRC used, when CRC use is enabled. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_CRC_TYPE_16</td><td>This selects the 16-bit polynomial CRC.</td></tr> <tr> <td>SIO4_HDLC_TX_CRC_TYPE_32</td><td>This selects the 32-bit polynomial CRC.</td></tr> <tr> <td>SIO4_HDLC_TX_CRC_TYPE_CCITT</td><td>This selects the 16-bit CCITT CRC. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_CRC_TYPE_16	This selects the 16-bit polynomial CRC.	SIO4_HDLC_TX_CRC_TYPE_32	This selects the 32-bit polynomial CRC.	SIO4_HDLC_TX_CRC_TYPE_CCITT	This selects the 16-bit CCITT CRC. This is the default.		
Value	Description										
SIO4_HDLC_TX_CRC_TYPE_16	This selects the 16-bit polynomial CRC.										
SIO4_HDLC_TX_CRC_TYPE_32	This selects the 32-bit polynomial CRC.										
SIO4_HDLC_TX_CRC_TYPE_CCITT	This selects the 16-bit CCITT CRC. This is the default.										
tx. crc. preset	<p>This field selects the CRC starting value. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_CRC_PRESET_ALL_0</td><td>Use a starting value of all zeroes.</td></tr> <tr> <td>SIO4_HDLC_TX_CRC_PRESET_ALL_1</td><td>Use a starting value of all ones. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_CRC_PRESET_ALL_0	Use a starting value of all zeroes.	SIO4_HDLC_TX_CRC_PRESET_ALL_1	Use a starting value of all ones. This is the default.				
Value	Description										
SIO4_HDLC_TX_CRC_PRESET_ALL_0	Use a starting value of all zeroes.										
SIO4_HDLC_TX_CRC_PRESET_ALL_1	Use a starting value of all ones. This is the default.										
tx. crc. on_end	<p>This field specifies if a CRC is to be send at the end of a Frame. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_CRC_ON_END_NO</td><td>Do <u>not</u> send a CRC.</td></tr> <tr> <td>SIO4_HDLC_TX_CRC_ON_END_YES</td><td>Do send a CRC. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_CRC_ON_END_NO	Do <u>not</u> send a CRC.	SIO4_HDLC_TX_CRC_ON_END_YES	Do send a CRC. This is the default.				
Value	Description										
SIO4_HDLC_TX_CRC_ON_END_NO	Do <u>not</u> send a CRC.										
SIO4_HDLC_TX_CRC_ON_END_YES	Do send a CRC. This is the default.										
tx. parity	This structure configures the transmitter's use of Parity checking.										
tx. parity. enable	<p>This field enables or disables the use of Parity. When enabled, the character size is inclusive of the Parity Bit, except in the size specified for the last character of the Frame. When used, the Parity Bit appears to the immediate left of the most significant data bit. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_PARITY_ENABLE_NO</td><td>Do <u>not</u> generate a Parity bit. This is the default.</td></tr> <tr> <td>SIO4_HDLC_TX_PARITY_ENABLE_YES</td><td>Do generate a Parity bit.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_PARITY_ENABLE_NO	Do <u>not</u> generate a Parity bit. This is the default.	SIO4_HDLC_TX_PARITY_ENABLE_YES	Do generate a Parity bit.				
Value	Description										
SIO4_HDLC_TX_PARITY_ENABLE_NO	Do <u>not</u> generate a Parity bit. This is the default.										
SIO4_HDLC_TX_PARITY_ENABLE_YES	Do generate a Parity bit.										
tx. parity. type	<p>This field specifies the type of Parity to use, when its use is enabled. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_PARITY_TYPE_EVEN</td><td>This specifies Even Parity. This is the default.</td></tr> <tr> <td>SIO4_HDLC_TX_PARITY_TYPE_ODD</td><td>This specifies Odd Parity.</td></tr> <tr> <td>SIO4_HDLC_TX_PARITY_TYPE_ONE</td><td>This specifies One Parity (the parity bit is always set).</td></tr> <tr> <td>SIO4_HDLC_TX_PARITY_TYPE_ZERO</td><td>This specifies Zero Parity (the parity bit is always clear).</td></tr> </table>	Value	Description	SIO4_HDLC_TX_PARITY_TYPE_EVEN	This specifies Even Parity. This is the default.	SIO4_HDLC_TX_PARITY_TYPE_ODD	This specifies Odd Parity.	SIO4_HDLC_TX_PARITY_TYPE_ONE	This specifies One Parity (the parity bit is always set).	SIO4_HDLC_TX_PARITY_TYPE_ZERO	This specifies Zero Parity (the parity bit is always clear).
Value	Description										
SIO4_HDLC_TX_PARITY_TYPE_EVEN	This specifies Even Parity. This is the default.										
SIO4_HDLC_TX_PARITY_TYPE_ODD	This specifies Odd Parity.										
SIO4_HDLC_TX_PARITY_TYPE_ONE	This specifies One Parity (the parity bit is always set).										
SIO4_HDLC_TX_PARITY_TYPE_ZERO	This specifies Zero Parity (the parity bit is always clear).										
tx. preamble	This structure configures the transmitter's use of a Preamble sequence, which is driven on the cable's Tx Data signal preceding each Frame. The Preamble can be used to force a minimum time delay between successive Frames.										
tx. preamble. enable	<p>This field enables or disables the use of a Preamble sequence. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_TX_PREAMBLE_ENABLE_NO</td><td>Do <u>not</u> send a Preamble sequence. This is the default.</td></tr> <tr> <td>SIO4_HDLC_TX_PREAMBLE_ENABLE_YES</td><td>Do send a Preamble sequence.</td></tr> </table>	Value	Description	SIO4_HDLC_TX_PREAMBLE_ENABLE_NO	Do <u>not</u> send a Preamble sequence. This is the default.	SIO4_HDLC_TX_PREAMBLE_ENABLE_YES	Do send a Preamble sequence.				
Value	Description										
SIO4_HDLC_TX_PREAMBLE_ENABLE_NO	Do <u>not</u> send a Preamble sequence. This is the default.										
SIO4_HDLC_TX_PREAMBLE_ENABLE_YES	Do send a Preamble sequence.										
tx. preamble. flag	This field enables or disables the use of the Flag sequence as the Preamble Pattern. This selection works in conjunction with the Preamble Pattern selection below. Valid values are given in the table below.										

	Value	Description
	SIO4_HDLC_TX_PREAMBLE_FLAG_NO	Do <u>not</u> use the Flag sequence.
	SIO4_HDLC_TX_PREAMBLE_FLAG_YES	<u>Do</u> use the Flag sequence. This is the default.
tx. preamble. pattern	This field selects the Preamble Pattern to use, when use of a Preamble is enabled. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_TX_PREAMBLE_PATTERN_0	This specifies continuous zero bits.
	SIO4_HDLC_TX_PREAMBLE_PATTERN_1	This specifies continuous one bits. If the above Flag option is Yes, then this option refers to the Flag sequence. This is the default.
	SIO4_HDLC_TX_PREAMBLE_PATTERN_01	This specifies a pattern of a zero bit followed by a one bit.
	SIO4_HDLC_TX_PREAMBLE_PATTERN_10	This specifies a pattern of a one bit followed by a zero bit.
tx. preamble. length	This field specifies the length of the Preamble Pattern to use, when use of a Preamble is enabled. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_TX_PREAMBLE_LENGTH_8_BITS	The length is 8-bits. This is the default.
	SIO4_HDLC_TX_PREAMBLE_LENGTH_16_BITS	The length is 16-bits.
	SIO4_HDLC_TX_PREAMBLE_LENGTH_32_BITS	The length is 32-bits.
	SIO4_HDLC_TX_PREAMBLE_LENGTH_64_BITS	The length is 64-bits.
tx. fifo	This structure configures the transmitter's FIFO parameters.	
tx. fifo. size	This field is filled in by the <code>sio4_hdlc_init_data()</code> call (section 4.1.3, page 12) with the size of the channel's Tx FIFO. This is offered for informational purposes only.	
tx. fifo. ae	This field specifies the Tx FIFO Almost Empty setting. The Tx FIFO Almost Empty status is asserted (goes low) when the Tx FIFO contains this number of values, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7.	
tx. fifo. af	This field specifies the Tx FIFO Almost Full setting. The Tx FIFO Almost Full status is asserted (goes low) when the Tx FIFO contains this number of free spaces, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7.	
tx. fifo. empty_cfg	This field configures the transmitter's reaction to the Tx FIFO becoming empty. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_TX_FIFO_EMPTY_CFG_IGNORE	This specifies that the condition is to be ignored. This is the default.
	SIO4_HDLC_TX_FIFO_EMPTY_CFG_TX_OFF	This specifies that the transmitter be disabled when the condition occurs.
tx. fifo. space_cfg	This field configures the FIFO space allocation between the transmitter and the receiver when the Tx FIFO and Rx FIFO are of different sizes. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_TX_FIFO_SPACE_CFG_RX_2X	This specifies that the Rx FIFO be twice as large as the Tx FIFO. This is the default.
	SIO4_HDLC_TX_FIFO_SPACE_CFG_TX_2X	This specifies that the Tx FIFO be twice as large as the Rx FIFO.

tx. io	This structure configures the transmitter's software settings. These settings are used during <code>sio4_hdlc_tx_frame()</code> calls (see section 4.1.14, page 22).								
tx. io. dma_thresh	This field configures the minimum size of DMA transfers to be performed by the write service. If the Tx FIFO has less than the specified amount of available space, then the driver will wait a single system timer interval before trying again. However, if the Tx FIFO can accommodate what remains of the current request, or if the output stream appears to be idle, then the driver will perform a transfer rather than wait for more space. The valid range is any non-negative value up to the size of the Tx FIFO. The default is 0. Refer to the Tx I/O DMA Threshold setting and "Operating Information" section of the <i>SIO4 Reference Manual</i> for additional information.								
tx. io. mode	<p>This field configures the mechanism used to transfer data from host memory to the channel's Tx FIFO. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td><code>SIO4_HDLC_TX_IO_MODE_BMDMA</code></td><td>This selects Block Mode DMA transfers. *</td></tr> <tr> <td><code>SIO4_HDLC_TX_IO_MODE_DMDMA</code></td><td>This selects Demand Mode DMA transfers. *</td></tr> <tr> <td><code>SIO4_HDLC_TX_IO_MODE_PIO</code></td><td>This selects PIO mode transfers. This is the default.</td></tr> </table> <p>* The SIO4 has only two DMA engines. A BMDMA or DMDMA transfer request will fail if both DMA engines are already in use by other SIO4 channels.</p>	Value	Description	<code>SIO4_HDLC_TX_IO_MODE_BMDMA</code>	This selects Block Mode DMA transfers. *	<code>SIO4_HDLC_TX_IO_MODE_DMDMA</code>	This selects Demand Mode DMA transfers. *	<code>SIO4_HDLC_TX_IO_MODE_PIO</code>	This selects PIO mode transfers. This is the default.
Value	Description								
<code>SIO4_HDLC_TX_IO_MODE_BMDMA</code>	This selects Block Mode DMA transfers. *								
<code>SIO4_HDLC_TX_IO_MODE_DMDMA</code>	This selects Demand Mode DMA transfers. *								
<code>SIO4_HDLC_TX_IO_MODE_PIO</code>	This selects PIO mode transfers. This is the default.								
tx. io. pio_thresh	This field specifies the threshold for write request sizes that force the use of PIO mode. If a write request is this size or less, then the transfer will automatically use PIO. The valid range is any non-negative value. The default is 44.								
tx. io. timeout	This field specifies the maximum duration of Tx Frame requests. This refers to various steps within the HDLC Protocol Library. The valid range is from zero to 3600. The units are seconds. The default is 10 seconds.								
tx. io. overrun	<p>This field tells the driver if it is to check for Tx FIFO overrun conditions before proceeding with write requests. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td><code>SIO4_HDLC_TX_IO_OVERRUN_CHECK</code></td><td>This specifies that the driver <u>should</u> check for overrun conditions.</td></tr> <tr> <td><code>SIO4_HDLC_TX_IO_OVERRUN_IGNORE</code></td><td>This specifies that the driver should <u>not</u> check for overrun conditions. This is the default. Overrun testing is performed by the library and need not be performed by the driver.</td></tr> </table>	Value	Description	<code>SIO4_HDLC_TX_IO_OVERRUN_CHECK</code>	This specifies that the driver <u>should</u> check for overrun conditions.	<code>SIO4_HDLC_TX_IO_OVERRUN_IGNORE</code>	This specifies that the driver should <u>not</u> check for overrun conditions. This is the default. Overrun testing is performed by the library and need not be performed by the driver.		
Value	Description								
<code>SIO4_HDLC_TX_IO_OVERRUN_CHECK</code>	This specifies that the driver <u>should</u> check for overrun conditions.								
<code>SIO4_HDLC_TX_IO_OVERRUN_IGNORE</code>	This specifies that the driver should <u>not</u> check for overrun conditions. This is the default. Overrun testing is performed by the library and need not be performed by the driver.								

4.2.1.4. `sio4_hdlc_t.rx`

This section describes the structure's receiver configuration fields.

Field	Description				
rx	This structure configures the receiver portion of the channel.				
rx. mode	<p>This field specifies the receiver's operating mode. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td><code>SIO4_HDLC_RX_MODE_HDLC</code></td><td>This selects the HDLC operating mode. This is the default and the only valid option for this library.</td></tr> </table>	Value	Description	<code>SIO4_HDLC_RX_MODE_HDLC</code>	This selects the HDLC operating mode. This is the default and the only valid option for this library.
Value	Description				
<code>SIO4_HDLC_RX_MODE_HDLC</code>	This selects the HDLC operating mode. This is the default and the only valid option for this library.				
rx. adrs	This specifies the desired receiver/station address. The receiver will ignore Frames addressed to other stations. Valid values are from zero to 0xFF. The default is 0xFF.				
rx. adrs_ctrl	<p>This field specifies the structure of the address and control fields expected in received Frames. The SIO4 hardware performs address comparison only against the first byte of the address and control field. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> </table>	Value	Description		
Value	Description				

	SIO4_HDLC_RX_ADRS_CTRL_A1_C1	This specifies a one byte Address field and a one byte Control field.
	SIO4_HDLC_RX_ADRS_CTRL_A1_C2	This specifies a one byte Address field and a two byte Control field.
	SIO4_HDLC_RX_ADRS_CTRL_A1_C3	This specifies a one byte Address field and a three byte Control field.
	SIO4_HDLC_RX_ADRS_CTRL_AE_1_CE	This specifies an Extended Address field and an Extended Control field, with a one byte field between them.
	SIO4_HDLC_RX_ADRS_CTRL_AE_2_CE	This specifies an Extended Address field and an Extended Control field, with a two byte field between them.
	SIO4_HDLC_RX_ADRS_CTRL_AE_C2	This specifies an Extended Address field and a two byte Control field.
	SIO4_HDLC_RX_ADRS_CTRL_AE_C3	This specifies an Extended Address field and a three byte Control field.
	SIO4_HDLC_RX_ADRS_CTRL_OFF	This disables Address and Control field detection. This is the default.
rx. enable	This field specifies if the receiver is to be enabled. When configuration is begun (see <code>sio4_hdlc_set()</code> , section 4.1.9, page 19) the receiver is initialized and disabled. The option in this field is applied towards the end of the configuration process. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_ENABLE_NO_AFTER	This disables the receiver after it has finished the reception in progress.
	SIO4_HDLC_RX_ENABLE_NO_NOW	This disables the receiver immediately.
	SIO4_HDLC_RX_ENABLE_YES_NOW	This enables the receiver immediately. This is the default.
	SIO4_HDLC_RX_ENABLE_YES_W_AE	This enables the receiver according to the state of any hardware flow control lines.
rx. char_len	This field specifies if the size of receiver characters. The length specified <u>includes</u> the Parity Bit, if Parity is enabled. The data bits are the lower significant bits of the byte. (Refers to the Z16C30 data book for exceptions.) Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_CHAR_LEN_1	Characters are 1-bit in length.
	SIO4_HDLC_RX_CHAR_LEN_2	Characters are 2-bits in length.
	SIO4_HDLC_RX_CHAR_LEN_3	Characters are 3-bits in length.
	SIO4_HDLC_RX_CHAR_LEN_4	Characters are 4-bits in length.
	SIO4_HDLC_RX_CHAR_LEN_5	Characters are 5-bits in length.
	SIO4_HDLC_RX_CHAR_LEN_6	Characters are 6-bits in length.
	SIO4_HDLC_RX_CHAR_LEN_7	Characters are 7-bits in length.
	SIO4_HDLC_RX_CHAR_LEN_8	Characters are 8-bits in length. This is the default.
rx. encoding	This field specifies the encoding of the received data. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_ENCODING_BI_MARK	This refers to Biphase Mark encoding.
	SIO4_HDLC_RX_ENCODING_BI_LEVEL	This refers to Biphase Level encoding.
	SIO4_HDLC_RX_ENCODING_BI_SPACE	This refers to Biphase Space encoding.
	SIO4_HDLC_RX_ENCODING_D_BI_LEVEL	This refers to Differential Biphase Level encoding.
	SIO4_HDLC_RX_ENCODING_NRZ	This refers to NRZ encoding.
	SIO4_HDLC_RX_ENCODING_NRZB	This refers to NRZB encoding.
	SIO4_HDLC_RX_ENCODING_NRZI_MARK	This refers to NRZI-Mark encoding.

	SIO4_HDLC_RX_ENCODING_NRZI_SPACE	This refers to NRZI-Space encoding. This is the default.
rx. size_limit	This specifies the maximum size of receive frames. The maximum is the default of 0xFFFF. At this time the HDLC Protocol Library does not support any value other than 0xFFFF.	
rx. bit_rate	This specifies the desired receive data bit rate. During the sio4_hdlc_init() call (section 4.1.3, page 12) this is computed from the sio4_hdlc_init_t.rx_bit_rate field provided to the call.	
rx. queue_abort	This field specifies if received Abort sequences are queued through the USC's Rx FIFO with the data. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_QUEUE_ABORT_NO	Retain the Abort received status, but do not queue it through the USC Rx FIFO. *
	SIO4_HDLC_RX_QUEUE_ABORT_YES	Queue the Abort received status through the USC Rx FIFO. This is the default. †
* The Abort received status is available even if it isn't queued through the USC Rx FIFO. In this case the status may be reported out of sync with the data received.		
† If the Abort received status is queued through the USC Rx FIFO with the data, then it inhibits the queuing of any Parity Error status. In this case, the Parity Error status is lost.		
rx. sync_byte	This specifies the value to be compared to received data as the data enters the Rx FIFO (the one outside the USC). This comparison can be used for interrupt generation. Valid values are from zero to 0xFF. The default is zero.	
rx. status_word	This field controls whether the firmware will place the USC Receive Control/Status Register in the Rx FIFO along with the received data. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_STATUS_WORD_DISABLE	The RCSR is <u>not</u> placed in the Rx FIFO. This is the default.
	SIO4_HDLC_RX_STATUS_WORD_ENABLE	The RCSR <u>is</u> placed in the Rx FIFO. *
* The HDLC Protocol Library does not accommodate reading Frames with the feature enabled. Application will have to develop their own frame reading mechanism in this case.		
rx. crc	This structure configures the receiver's use of a CRC at the end of a Frame.	
rx. crc. enable	This field enables or disables use of a CRC at the end of frames. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_CRC_ENABLE_NO	CRCs are <u>not</u> used.
	SIO4_HDLC_RX_CRC_ENABLE_YES	CRCs <u>are</u> used. This is the default.
rx. crc. type	This field selects the type of CRC used, when CRC use is enabled. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_CRC_TYPE_16	This selects the 16-bit polynomial CRC.
	SIO4_HDLC_RX_CRC_TYPE_32	This selects the 32-bit polynomial CRC.
	SIO4_HDLC_RX_CRC_TYPE_CCITT	This selects the 16-bit CCITT CRC. This is the default.
rx. crc. preset	This field selects the CRC starting value. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_CRC_PRESET_ALL_0	Use a starting value of all zeroes.
	SIO4_HDLC_RX_CRC_PRESET_ALL_1	Use a starting value of all ones. This is the default.
rx. parity	This structure configures the receiver's use of Parity checking.	

rx. parity. enable	This field enables or disables the use of Parity. When enabled, the character size is inclusive of the Parity Bit, except in the size specified for the last character of the Frame. When used, the Parity Bit appears to the immediate left of the most significant data bit. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_PARITY_ENABLE_NO	Do <u>not</u> generate a Parity bit. This is the default.
	SIO4_HDLC_RX_PARITY_ENABLE_YES	<u>Do</u> generate a Parity bit.
rx. parity. type	This field specifies the type of Parity to use, when its use is enabled. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_PARITY_TYPE_EVEN	This specifies Even Parity. This is the default.
	SIO4_HDLC_RX_PARITY_TYPE_ODD	This specifies Odd Parity.
	SIO4_HDLC_RX_PARITY_TYPE_ONE	This specifies One Parity (the parity bit is always set).
	SIO4_HDLC_RX_PARITY_TYPE_ZERO	This specifies Zero Parity (the parity bit is always clear).
rx. fifo	This structure configures the receiver's FIFO parameters.	
rx. fifo. size	This field is filled in by the <code>sio4_hdlc_init_data()</code> call (section 4.1.3, page 12) with the size of the channel's Rx FIFO. This is offered for informational purposes only.	
rx. fifo. ae	This field specifies the Rx FIFO Almost Empty setting. The Rx FIFO Almost Empty status is asserted (goes low) when the Rx FIFO contain this number of values, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7.	
rx. fifo. af	This field specifies the Rx FIFO Almost Full setting. The Rx FIFO Almost Full status is asserted (goes low) when the Rx FIFO contain this number of free spaces, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7.	
rx. fifo. full_cfg	This field configures the receiver's reaction to the Rx FIFO becoming full. Valid values are given in the table below. This field refers to the channel specific setting, when supported. The corresponding global setting is not handled by the HDLC Protocol Library. The global setting must be handled separately by the application.	
	Value	Description
	SIO4_HDLC_RX_FIFO_FULL_CFG_DISABLE	This specifies that the receiver be disabled when the condition occurs.
	SIO4_HDLC_RX_FIFO_FULL_CFG_OVER	This specifies that the condition be ignored. This is the default.
rx. io	This structure configures the receiver's software settings. These settings are used during <code>sio4_hdlc_rx_frame()</code> calls (section 4.1.8, page 16).	
rx. io. dma_thresh	This field specifies the minimum size of DMA transfers when the driver needs to wait for additional data to become available in the Rx FIFO. If data is available in the Rx FIFO, but it is less than the specified threshold and won't fulfill the request, then the driver will wait for additional data to become available. The wait period is one system timer tick. The valid range is any non-negative value up to the size of the Rx FIFO. The default is 0. Refer to the Rx I/O DMA Threshold setting and "Operating Information" section of the <i>SIO4 Reference Manual</i> for additional information.	
rx. io. mode	This field configures the mechanism used to transfer data from the channel's Rx FIFO to host memory. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_IO_MODE_BMDMA	This selects Block Mode DMA transfers. *
	SIO4_HDLC_RX_IO_MODE_DMDMA	This selects Demand Mode DMA transfers. *
	SIO4_HDLC_RX_IO_MODE_PIO	This selects PIO mode transfers. This is the default.

	* The SIO4 has only two DMA engines. A BMDMA or DMDMA transfer request will fail if both DMA engines are already in use by other SIO4 channels.	
rx. io. pio_thresh	This field specifies the threshold for read request sizes that force the use of PIO mode. If a read request is this size or less, then the transfer will automatically use PIO. The valid range is any non-negative value. The default is 44.	
rx. io. timeout	This field specifies the maximum duration of Rx Frame requests. This refers to various steps within the HDLC Protocol Library. The valid range is from zero to 3600. The units are seconds. The default is 10 seconds.	
rx. io. overrun	This field tells the driver if it is to check for Rx FIFO overrun conditions before proceeding with read requests. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_IO_OVERRUN_CHECK	This specifies that the driver <u>should</u> check for overrun conditions.
rx. io. underrun	This field tells the driver if it is to check for Rx FIFO underrun conditions before proceeding with read requests. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_IO_UNDERRUN_CHECK	This specifies that the driver <u>should</u> check for underrun conditions.
rx. time_stamp	This structure configures the receiver's Time Stamp settings.	
	This field enables or disables the channels use of the Time Stamp feature. Valid values are given in the table below.	
	Value	Description
rx. time_stamp. enable	SIO4_HDLC_RX_TIME_STAMP_ENABLE_NO	Do <u>not</u> use the Time Stamp feature. This is the default.
	SIO4_HDLC_RX_TIME_STAMP_ENABLE_YES	<u>Do</u> use the Time Stamp feature. *
	* The HDLC Protocol Library does not accommodate reading Frames with this enabled.	
rx. time_stamp. clk_src	This field selects the Time Stamp clock source. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_RX_TIME_STAMP_CLK_SRC_EXT	Use the board's external TTL clock source. *
	SIO4_HDLC_RX_TIME_STAMP_CLK_SRC_INT	Use the board's internal 1us clock. This is the default. *
	* All four channels on the SIO4 use the same clock source.	

4.2.1.5. sio4_hdlc_t.usc

This section describes the structure's USC configuration fields.

Field	Description												
usc	This structure configures the USC portion of the channel. These fields are filled according to the bit rates requested for the transmitter and the receiver, and the receiver's use, or not, of the cable's Rx Clock signal.												
usc. mode	<p>This field specifies the USC's overall operating mode. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_MODE_AUTO_ECHO</td><td>This is the USC's Auto Echo mode.</td></tr> <tr> <td>SIO4_HDLC_USC_MODE_LOOPBACK_EXT</td><td>This is the USC's external loopback mode.</td></tr> <tr> <td>SIO4_HDLC_USC_MODE_LOOPBACK_INT</td><td>This is the USC's internal loopback mode.</td></tr> <tr> <td>SIO4_HDLC_USC_MODE_NORMAL</td><td>This is the USC's normal operating mode. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_USC_MODE_AUTO_ECHO	This is the USC's Auto Echo mode.	SIO4_HDLC_USC_MODE_LOOPBACK_EXT	This is the USC's external loopback mode.	SIO4_HDLC_USC_MODE_LOOPBACK_INT	This is the USC's internal loopback mode.	SIO4_HDLC_USC_MODE_NORMAL	This is the USC's normal operating mode. This is the default.		
Value	Description												
SIO4_HDLC_USC_MODE_AUTO_ECHO	This is the USC's Auto Echo mode.												
SIO4_HDLC_USC_MODE_LOOPBACK_EXT	This is the USC's external loopback mode.												
SIO4_HDLC_USC_MODE_LOOPBACK_INT	This is the USC's internal loopback mode.												
SIO4_HDLC_USC_MODE_NORMAL	This is the USC's normal operating mode. This is the default.												
usc. txd	<p>This field configures the operation of the USC's Tx Data pin. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_TXD_OUT_0</td><td>The pin is driven low.</td></tr> <tr> <td>SIO4_HDLC_USC_TXD_OUT_1</td><td>The pin is driven high.</td></tr> <tr> <td>SIO4_HDLC_USC_TXD_OUT_TXD</td><td>The pin is driven from the transmitter's Tx Data signal. This is the default.</td></tr> <tr> <td>SIO4_HDLC_USC_TXD_TRI</td><td>The pin is tri-stated.</td></tr> </table>	Value	Description	SIO4_HDLC_USC_TXD_OUT_0	The pin is driven low.	SIO4_HDLC_USC_TXD_OUT_1	The pin is driven high.	SIO4_HDLC_USC_TXD_OUT_TXD	The pin is driven from the transmitter's Tx Data signal. This is the default.	SIO4_HDLC_USC_TXD_TRI	The pin is tri-stated.		
Value	Description												
SIO4_HDLC_USC_TXD_OUT_0	The pin is driven low.												
SIO4_HDLC_USC_TXD_OUT_1	The pin is driven high.												
SIO4_HDLC_USC_TXD_OUT_TXD	The pin is driven from the transmitter's Tx Data signal. This is the default.												
SIO4_HDLC_USC_TXD_TRI	The pin is tri-stated.												
usc. cts	<p>This field configures the operation of the USC's CTS pin. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_CTS_OUT_0</td><td>The pin is driven low.</td></tr> <tr> <td>SIO4_HDLC_USC_CTS_OUT_1</td><td>The pin is driven high.</td></tr> <tr> <td>SIO4_HDLC_USC_CTS_IN_CBL_CTS</td><td>The pin is an input driver from the cable's CTS signal.</td></tr> <tr> <td>SIO4_HDLC_USC_CTS_TRI</td><td>The pin is tri-stated. This is the default.</td></tr> </table>	Value	Description	SIO4_HDLC_USC_CTS_OUT_0	The pin is driven low.	SIO4_HDLC_USC_CTS_OUT_1	The pin is driven high.	SIO4_HDLC_USC_CTS_IN_CBL_CTS	The pin is an input driver from the cable's CTS signal.	SIO4_HDLC_USC_CTS_TRI	The pin is tri-stated. This is the default.		
Value	Description												
SIO4_HDLC_USC_CTS_OUT_0	The pin is driven low.												
SIO4_HDLC_USC_CTS_OUT_1	The pin is driven high.												
SIO4_HDLC_USC_CTS_IN_CBL_CTS	The pin is an input driver from the cable's CTS signal.												
SIO4_HDLC_USC_CTS_TRI	The pin is tri-stated. This is the default.												
usc. cts_legacy	<p>This field configures the operation of the USC's CTS pin for legacy mode cable interface configurations. Valid values are given in the table below. Please also read Cable Configuration Modes (section 5.10, page 61).</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_CTS_LEG_IN</td><td>The pin operates as an input. This is the default.</td></tr> <tr> <td>SIO4_HDLC_USC_CTS_LEG_OUT_0</td><td>The pin operates as an output driven low.</td></tr> <tr> <td>SIO4_HDLC_USC_CTS_LEG_OUT_1</td><td>The pin operates as an output driven high.</td></tr> </table>	Value	Description	SIO4_HDLC_USC_CTS_LEG_IN	The pin operates as an input. This is the default.	SIO4_HDLC_USC_CTS_LEG_OUT_0	The pin operates as an output driven low.	SIO4_HDLC_USC_CTS_LEG_OUT_1	The pin operates as an output driven high.				
Value	Description												
SIO4_HDLC_USC_CTS_LEG_IN	The pin operates as an input. This is the default.												
SIO4_HDLC_USC_CTS_LEG_OUT_0	The pin operates as an output driven low.												
SIO4_HDLC_USC_CTS_LEG_OUT_1	The pin operates as an output driven high.												
usc. dcd	<p>This field configures the operation of the USC's DCD pin. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_DCD_DISABLE</td><td>The pin is disabled. This is the default.</td></tr> <tr> <td>SIO4_HDLC_USC_DCD_IN_DCD_CBL_DCD</td><td>The pin is an input for the receiver's DCD function and is driven from the cable's DCD signal.</td></tr> <tr> <td>SIO4_HDLC_USC_DCD_IN_SYNC_CBL_DCD</td><td>The pin is an input for the receiver's SYNC function and is driven from the cable's DCD signal.</td></tr> <tr> <td>SIO4_HDLC_USC_DCD_OUT_0</td><td>The pin is driven low. *</td></tr> <tr> <td>SIO4_HDLC_USC_DCD_OUT_1</td><td>The pin is driven high. *</td></tr> </table> <p>* This option enables the cable DCD signal to be driven, though the cable.dcd field (section 4.2.1.2, page 30) may configure the cable to output an alternate signal.</p>	Value	Description	SIO4_HDLC_USC_DCD_DISABLE	The pin is disabled. This is the default.	SIO4_HDLC_USC_DCD_IN_DCD_CBL_DCD	The pin is an input for the receiver's DCD function and is driven from the cable's DCD signal.	SIO4_HDLC_USC_DCD_IN_SYNC_CBL_DCD	The pin is an input for the receiver's SYNC function and is driven from the cable's DCD signal.	SIO4_HDLC_USC_DCD_OUT_0	The pin is driven low. *	SIO4_HDLC_USC_DCD_OUT_1	The pin is driven high. *
Value	Description												
SIO4_HDLC_USC_DCD_DISABLE	The pin is disabled. This is the default.												
SIO4_HDLC_USC_DCD_IN_DCD_CBL_DCD	The pin is an input for the receiver's DCD function and is driven from the cable's DCD signal.												
SIO4_HDLC_USC_DCD_IN_SYNC_CBL_DCD	The pin is an input for the receiver's SYNC function and is driven from the cable's DCD signal.												
SIO4_HDLC_USC_DCD_OUT_0	The pin is driven low. *												
SIO4_HDLC_USC_DCD_OUT_1	The pin is driven high. *												
usc. dcd_legacy	This field configures the operation of the USC's DCD pin for legacy mode cable interface configurations. Valid values are given in the table below. Please also read Cable Configuration												

	<p>Modes (section 5.10, page 61).</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_DCD_LEG_IN_DCD</td><td>The pin operates as a DCD input. This is the default.</td></tr> <tr> <td>SIO4_HDLC_USC_DCD_LEG_IN_SYNC</td><td>The pin operates as a SYNC input.</td></tr> <tr> <td>SIO4_HDLC_USC_DCD_LEG_OUT_0</td><td>The pin operates as an output driven low.</td></tr> <tr> <td>SIO4_HDLC_USC_DCD_LEG_OUT_1</td><td>The pin operates as an output driven high.</td></tr> </table>	Value	Description	SIO4_HDLC_USC_DCD_LEG_IN_DCD	The pin operates as a DCD input. This is the default.	SIO4_HDLC_USC_DCD_LEG_IN_SYNC	The pin operates as a SYNC input.	SIO4_HDLC_USC_DCD_LEG_OUT_0	The pin operates as an output driven low.	SIO4_HDLC_USC_DCD_LEG_OUT_1	The pin operates as an output driven high.																		
Value	Description																												
SIO4_HDLC_USC_DCD_LEG_IN_DCD	The pin operates as a DCD input. This is the default.																												
SIO4_HDLC_USC_DCD_LEG_IN_SYNC	The pin operates as a SYNC input.																												
SIO4_HDLC_USC_DCD_LEG_OUT_0	The pin operates as an output driven low.																												
SIO4_HDLC_USC_DCD_LEG_OUT_1	The pin operates as an output driven high.																												
usc.tx	This structure configures a few of the USC's transmitter settings.																												
usc.tx.clk_src	<p>This field configures the source for the USC transmitter clock. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_BRG0</td><td>Select Baud Rate Generator 0.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_BRG1</td><td>Select Baud Rate Generator 1.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_CTR0</td><td>Select Counter 0.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_CTR1</td><td>Select Counter 1.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_DISABLE</td><td>Disable the transmitter.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_DPLL</td><td>Select the DPLL.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_RXC_PIN</td><td>Select the Rx Clock pin.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_CLK_SRC_TXC_PIN</td><td>Select the Tx Clock pin. *</td></tr> </table> <p>* This is the initial default, though it may change to satisfy bit rate requirements.</p>	Value	Description	SIO4_HDLC_USC_TX_CLK_SRC_BRG0	Select Baud Rate Generator 0.	SIO4_HDLC_USC_TX_CLK_SRC_BRG1	Select Baud Rate Generator 1.	SIO4_HDLC_USC_TX_CLK_SRC_CTR0	Select Counter 0.	SIO4_HDLC_USC_TX_CLK_SRC_CTR1	Select Counter 1.	SIO4_HDLC_USC_TX_CLK_SRC_DISABLE	Disable the transmitter.	SIO4_HDLC_USC_TX_CLK_SRC_DPLL	Select the DPLL.	SIO4_HDLC_USC_TX_CLK_SRC_RXC_PIN	Select the Rx Clock pin.	SIO4_HDLC_USC_TX_CLK_SRC_TXC_PIN	Select the Tx Clock pin. *										
Value	Description																												
SIO4_HDLC_USC_TX_CLK_SRC_BRG0	Select Baud Rate Generator 0.																												
SIO4_HDLC_USC_TX_CLK_SRC_BRG1	Select Baud Rate Generator 1.																												
SIO4_HDLC_USC_TX_CLK_SRC_CTR0	Select Counter 0.																												
SIO4_HDLC_USC_TX_CLK_SRC_CTR1	Select Counter 1.																												
SIO4_HDLC_USC_TX_CLK_SRC_DISABLE	Disable the transmitter.																												
SIO4_HDLC_USC_TX_CLK_SRC_DPLL	Select the DPLL.																												
SIO4_HDLC_USC_TX_CLK_SRC_RXC_PIN	Select the Rx Clock pin.																												
SIO4_HDLC_USC_TX_CLK_SRC_TXC_PIN	Select the Tx Clock pin. *																												
usc.tx.txc	<p>This field configures the operation of the USC's Tx Clock pin. Valid values are given in the table below.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_IN_0</td><td>The pin is an input driven low.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_IN_1</td><td>The pin is an input driven high.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_IN_CBL_RXAUX</td><td>The pin is an input driven from the cable's Rx Aux signal.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_IN_CBL_RXC</td><td>The pin is an input driven from the cable's Rx Clock signal.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_IN_OSC</td><td>The pin is an input driven from the onboard oscillator.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_IN_OSC_INV</td><td>The pin is an input driven from the inverted onboard oscillator.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_OUT_BRG0</td><td>The pin is an output driven from Baud Rate Generator 0.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_OUT_BRG1</td><td>The pin is an output driven from Baud Rate Generator 1.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_OUT_BYTE_CLK</td><td>The pin is an output driven from the transmitter's Byte Clock.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_OUT_CLK</td><td>The pin is an output driven from the transmit clock. *</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_OUT_COMP</td><td>The pin is an output driven from the transmit complete signal.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_OUT_CTR1</td><td>The pin is an output driven from Counter 1.</td></tr> <tr> <td>SIO4_HDLC_USC_TX_TXC_OUT_DPLL_TX</td><td>The pin is an output driven from the transmit clock from the DPLL.</td></tr> </table> <p>* This is the initial default, though it may change to satisfy bit rate requirements.</p>	Value	Description	SIO4_HDLC_USC_TX_TXC_IN_0	The pin is an input driven low.	SIO4_HDLC_USC_TX_TXC_IN_1	The pin is an input driven high.	SIO4_HDLC_USC_TX_TXC_IN_CBL_RXAUX	The pin is an input driven from the cable's Rx Aux signal.	SIO4_HDLC_USC_TX_TXC_IN_CBL_RXC	The pin is an input driven from the cable's Rx Clock signal.	SIO4_HDLC_USC_TX_TXC_IN_OSC	The pin is an input driven from the onboard oscillator.	SIO4_HDLC_USC_TX_TXC_IN_OSC_INV	The pin is an input driven from the inverted onboard oscillator.	SIO4_HDLC_USC_TX_TXC_OUT_BRG0	The pin is an output driven from Baud Rate Generator 0.	SIO4_HDLC_USC_TX_TXC_OUT_BRG1	The pin is an output driven from Baud Rate Generator 1.	SIO4_HDLC_USC_TX_TXC_OUT_BYTE_CLK	The pin is an output driven from the transmitter's Byte Clock.	SIO4_HDLC_USC_TX_TXC_OUT_CLK	The pin is an output driven from the transmit clock. *	SIO4_HDLC_USC_TX_TXC_OUT_COMP	The pin is an output driven from the transmit complete signal.	SIO4_HDLC_USC_TX_TXC_OUT_CTR1	The pin is an output driven from Counter 1.	SIO4_HDLC_USC_TX_TXC_OUT_DPLL_TX	The pin is an output driven from the transmit clock from the DPLL.
Value	Description																												
SIO4_HDLC_USC_TX_TXC_IN_0	The pin is an input driven low.																												
SIO4_HDLC_USC_TX_TXC_IN_1	The pin is an input driven high.																												
SIO4_HDLC_USC_TX_TXC_IN_CBL_RXAUX	The pin is an input driven from the cable's Rx Aux signal.																												
SIO4_HDLC_USC_TX_TXC_IN_CBL_RXC	The pin is an input driven from the cable's Rx Clock signal.																												
SIO4_HDLC_USC_TX_TXC_IN_OSC	The pin is an input driven from the onboard oscillator.																												
SIO4_HDLC_USC_TX_TXC_IN_OSC_INV	The pin is an input driven from the inverted onboard oscillator.																												
SIO4_HDLC_USC_TX_TXC_OUT_BRG0	The pin is an output driven from Baud Rate Generator 0.																												
SIO4_HDLC_USC_TX_TXC_OUT_BRG1	The pin is an output driven from Baud Rate Generator 1.																												
SIO4_HDLC_USC_TX_TXC_OUT_BYTE_CLK	The pin is an output driven from the transmitter's Byte Clock.																												
SIO4_HDLC_USC_TX_TXC_OUT_CLK	The pin is an output driven from the transmit clock. *																												
SIO4_HDLC_USC_TX_TXC_OUT_COMP	The pin is an output driven from the transmit complete signal.																												
SIO4_HDLC_USC_TX_TXC_OUT_CTR1	The pin is an output driven from Counter 1.																												
SIO4_HDLC_USC_TX_TXC_OUT_DPLL_TX	The pin is an output driven from the transmit clock from the DPLL.																												
usc.tx.txc_legacy	This field configures the operation of the USC's Tx Clock pin for legacy mode cable interface configurations. Valid values are given in the table below. Please also read Cable Configuration																												

	Modes (section 5.10, page 61).	
	Value	Description
	SIO4_HDLC_USC_TX_TXC_LEG_IN	The pin operates as an input.
	SIO4_HDLC_USC_TX_TXC_LEG_OUT_BRG0	The pin is an output driven from Baud Rate Generator 0.
	SIO4_HDLC_USC_TX_TXC_LEG_OUT_BRG1	The pin is an output driven from Baud Rate Generator 1.
	SIO4_HDLC_USC_TX_TXC_LEG_OUT_BYTE_CLK	The pin is an output driven from the transmitter's Byte Clock.
	SIO4_HDLC_USC_TX_TXC_LEG_OUT_CLK	The pin is an output driven from the transmit clock. *
	SIO4_HDLC_USC_TX_TXC_LEG_OUT_COMP	The pin is an output driven from the transmit complete signal.
	SIO4_HDLC_USC_TX_TXC_LEG_OUT_CTR1	The pin is an output driven from Counter 1.
	SIO4_HDLC_USC_TX_TXC_LEG_OUT_DPLL_TX	The pin is an output driven from the transmit clock from the DPLL.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. rx	This structure configures a few of the USC's receiver settings.	
usc. rx. clk_src	This field configures the source for the USC receiver clock. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_RX_CLK_SRC_BRG0	Select Baud Rate Generator 0.
	SIO4_HDLC_USC_RX_CLK_SRC_BRG1	Select Baud Rate Generator 1.
	SIO4_HDLC_USC_RX_CLK_SRC_CTR0	Select Counter 0.
	SIO4_HDLC_USC_RX_CLK_SRC_CTR1	Select Counter 1.
	SIO4_HDLC_USC_RX_CLK_SRC_DISABLE	Disable the receiver.
	SIO4_HDLC_USC_RX_CLK_SRC_DPLL	Select the DPLL.
	SIO4_HDLC_USC_RX_CLK_SRC_RXC_PIN	Select the Rx Clock pin. *
	SIO4_HDLC_USC_RX_CLK_SRC_TXC_PIN	Select the Tx Clock pin.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. rx. rxc	This field configures the operation of the USC's Rx Clock pin. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_RX_RXC_IN_0	The pin is an input driven low.
	SIO4_HDLC_USC_RX_RXC_IN_1	The pin is an input driven high.
	SIO4_HDLC_USC_RX_RXC_IN_CBL_RXAUX	The pin is an input driven from the cable's Rx Aux signal.
	SIO4_HDLC_USC_RX_RXC_IN_CBL_RXC	The pin is an input driven from the cable's Rx Clock signal.
	SIO4_HDLC_USC_RX_RXC_IN_OSC	The pin is an input driven from the onboard oscillator. *
	SIO4_HDLC_USC_RX_RXC_IN_OSC_INV	The pin is an input driven from the inverted onboard oscillator.
	SIO4_HDLC_USC_RX_RXC_OUT_BRG0	The pin is an output driven from Baud Rate Generator 0.
	SIO4_HDLC_USC_RX_RXC_OUT_BRG1	The pin is an output driven from Baud Rate Generator 1.
	SIO4_HDLC_USC_RX_RXC_OUT_BYTE_CLK	The pin is an output driven from the receiver's Byte Clock.
	SIO4_HDLC_USC_RX_RXC_OUT_CLK	The pin is an output driven from the

		receiver clock.
	SIO4_HDLC_USC_RX_RXC_OUT_CTR0	The pin is an output driven from Counter 0.
	SIO4_HDLC_USC_RX_RXC_OUT_DPLL_RX	The pin is an output driven from the receive clock from the DPLL.
	SIO4_HDLC_USC_RX_RXC_OUT_SYNC	The pin is an output driven from input SYNC signal.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. rx. rxc_legacy	This field configures the operation of the USC's Rx Clock pin for legacy mode cable interface configurations. Valid values are given in the table below. Please also read Cable Configuration Modes (section 5.10, page 61).	
	Value	Description
	SIO4_HDLC_USC_RX_RXC_LEG_IN	The pin is an input. *
	SIO4_HDLC_USC_RX_RXC_LEG_OUT_BRG0	The pin is an output driven from Baud Rate Generator 0.
	SIO4_HDLC_USC_RX_RXC_LEG_OUT_BRG1	The pin is an output driven from Baud Rate Generator 1.
	SIO4_HDLC_USC_RX_RXC_LEG_OUT_BYTE_CLK	The pin is an output driven from the receiver's Byte Clock.
	SIO4_HDLC_USC_RX_RXC_LEG_OUT_CLK	The pin is an output driven from the receiver clock.
	SIO4_HDLC_USC_RX_RXC_LEG_OUT_CTR0	The pin is an output driven from Counter 0.
	SIO4_HDLC_USC_RX_RXC_LEG_OUT_DPLL_RX	The pin is an output driven from the receive clock from the DPLL.
	SIO4_HDLC_USC_RX_RXC_LEG_OUT_SYNC	The pin is an output driven from input SYNC signal.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. brg0	This structure configures a few of the settings for Baud Rate Generator 0 (BRG0).	
usc. brg0. enable	This field enables or disabled BRG0. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_BRG0_ENABLE_NO	This disables BRG0. *
	SIO4_HDLC_USC_BRG0_ENABLE_YES	This enables BRG0.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. brg0. clk_src	This field selects the clock source for BRG0. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_BRG0_CLK_SRC_CTR0	This selects the output from Counter 0. *
	SIO4_HDLC_USC_BRG0_CLK_SRC_CTR1	This selects the output from Counter 1.
	SIO4_HDLC_USC_BRG0_CLK_SRC_RXC_PIN	This selects the signal present at the USC's Rx Clock pin.
	SIO4_HDLC_USC_BRG0_CLK_SRC_TXC_PIN	This selects the signal present at the USC's Tx Clock pin.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. brg0. divider	This field specifies the clock divider value for BRG0. The valid value range is from zero to 0xFFFF. The initial default is zero, though it may change to satisfy bit rate requirements.	
usc. brg0. mode	This field specifies the BRG0 operating mode. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_BRG0_MODE_CONT	This selects continuous operation. This is the default.

	SIO4_HDLC_USC_BRG0_MODE_SINGLE	This selects single shot mode, in which clocking stops when the counter value reaches zero.
usc. brg1	This structure configures a few of the settings for Baud Rate Generator 1 (BRG1).	
usc. brg1. enable	This field enables or disabled BRG1. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_BRG1_ENABLE_NO	This disables BRG1. *
	SIO4_HDLC_USC_BRG1_ENABLE_YES	This enables BRG1.
	* This is the initial default, though it may change to satisfy bit rate requirements.	
usc. brg1. clk_src	This field selects the clock source for BRG1. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_BRG1_CLK_SRC_CTR0	This selects the output from Counter 0.
	SIO4_HDLC_USC_BRG1_CLK_SRC_CTR1	This selects the output from Counter 1. *
	SIO4_HDLC_USC_BRG1_CLK_SRC_RXC_PIN	This selects the signal present at the USC's Rx Clock pin.
	SIO4_HDLC_USC_BRG1_CLK_SRC_TXC_PIN	This selects the signal present at the USC's Tx Clock pin.
	* This is the initial default, though it may change to satisfy bit rate requirements.	
usc. brg1. divider	This field specifies the clock divider value for BRG1. The valid value range is from zero to 0xFFFF. The initial default is zero, though it may change to satisfy bit rate requirements.	
usc. brg1. mode	This field specifies the BRG1 operating mode. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_BRG1_MODE_CONT	This selects continuous operation. This is the default.
	SIO4_HDLC_USC_BRG1_MODE_SINGLE	This selects single shot mode, in which clocking stops when the counter value reaches zero.
usc. ctr0	This structure configures a few of the settings for Counter 0 (CTR0).	
usc. ctr0. clk_src	This field selects the clock source for CTR0. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_CTR0_CLK_SRC_DISABLE	This disables CTR0. *
	SIO4_HDLC_USC_CTR0_CLK_SRC_RXC_PIN	This selects the signal present at the USC's Rx Clock pin.
	SIO4_HDLC_USC_CTR0_CLK_SRC_TXC_PIN	This selects the signal present at the USC's Tx Clock pin.
		* This is the initial default, though it may change to satisfy bit rate requirements.
usc. ctr0. rate	This field selects the divider rate for CTR0. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_CTR0_RATE_4X	This sets the output as the input divided by four.
	SIO4_HDLC_USC_CTR0_RATE_8X	This sets the output as the input divided by eight.
	SIO4_HDLC_USC_CTR0_RATE_16X	This sets the output as the input divided by 16.
	SIO4_HDLC_USC_CTR0_RATE_32X	This sets the output as the input divided by 32. *
	* This is the initial default, though it may change to satisfy bit rate requirements.	
usc. ctrl	This structure configures a few of the settings for Counter 1 (CTR1).	
usc. ctrl.	This field selects the clock source for CTR1. Valid values are given in the table below.	

clk_src	Value	Description
	SIO4_HDLC_USC_CTR1_CLK_SRC_DISABLE	This disables CTR1. *
	SIO4_HDLC_USC_CTR1_CLK_SRC_RXC_PIN	This selects the signal present at the USC's Rx Clock pin.
	SIO4_HDLC_USC_CTR1_CLK_SRC_TXC_PIN	This selects the signal present at the USC's Tx Clock pin.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. ctrl. rate_src	This field selects the source for the rate divider used by CTR1. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_CTR1_RATE_SRC_CTR0	This selects the rate divider used by CTR0.
	SIO4_HDLC_USC_CTR1_RATE_SRC_DPLL	This selects the rate divider used by the DPLL. *
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. dpll	This structure configures a few of the settings for the DPLL.	
usc. dpll. clk_src	This field selects the clock source for the DPLL. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_DPLL_CLK_SRC_BRG0	This selects the output from BRG0. *
	SIO4_HDLC_USC_DPLL_CLK_SRC_BRG1	This selects the output from BRG1.
	SIO4_HDLC_USC_DPLL_CLK_SRC_RXC_PIN	This selects the signal present at the USC's Rx Clock pin.
	SIO4_HDLC_USC_DPLL_CLK_SRC_TXC_PIN	This selects the signal present at the USC's Tx Clock pin.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. dpll. mode	This field specifies the DPLL operating mode, which corresponds to the Rx Data Encoding format. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_DPLL_MODE_BIPH_LVL	This refers to Biphase-Level.
	SIO4_HDLC_USC_DPLL_MODE_BIPH_MS	This refers to either Biphase-Mark or Biphase Space.
	SIO4_HDLC_USC_DPLL_MODE_DISABLE	This disables the DPLL. *
	SIO4_HDLC_USC_DPLL_MODE_NRZ_NRZI	This refers to either NRZ or NRZI.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. dpll. rate	This field selects the divider rate for the DPLL. Valid values are given in the table below.	
	Value	Description
	SIO4_HDLC_USC_DPLL_RATE_CTR1_4X	This option cannot be used if the DPLL is utilized. This is a divide by four option, but can be selected only if the DPLL is the source for CTR1's rate divider value.
	SIO4_HDLC_USC_DPLL_RATE_8X	This sets the output as the input divided by eight. *
	SIO4_HDLC_USC_DPLL_RATE_16X	This sets the output as the input divided by 16.
	SIO4_HDLC_USC_DPLL_RATE_32X	This sets the output as the input divided by 32.
* This is the initial default, though it may change to satisfy bit rate requirements.		
usc. dpll. edge	This field selects the source signal edges that the DPLL uses for synchronization. Valid values are given in the table below.	
	Value	Description

	SIO4_HDLC_USC_DPLL_EDGE_BOTH_EDGE	This selects both rising and falling edges that the DPLL is to use for clocking synchronization.
	SIO4_HDLC_USC_DPLL_EDGE_FALL_EDGE	This selects the falling edges that the DPLL is to use for clocking synchronization.
	SIO4_HDLC_USC_DPLL_EDGE_INHIBIT	This inhibits the DPLL from synchronizing on the input clock. *
	SIO4_HDLC_USC_DPLL_EDGE_RISE_EDGE	This selects the rising edges that the DPLL is to use for clocking synchronization.
* This is the initial default, though it may change to satisfy bit rate requirements.		

5. Operating Information

5.1. Basic Illustration

The below figure is included to assist individuals in the configuration of the SIO4. The figure illustrates boards with more recent firmware. The DMA references are handled automatically by the driver to facilitate movement of data between the USC and the on-board FIFOs.

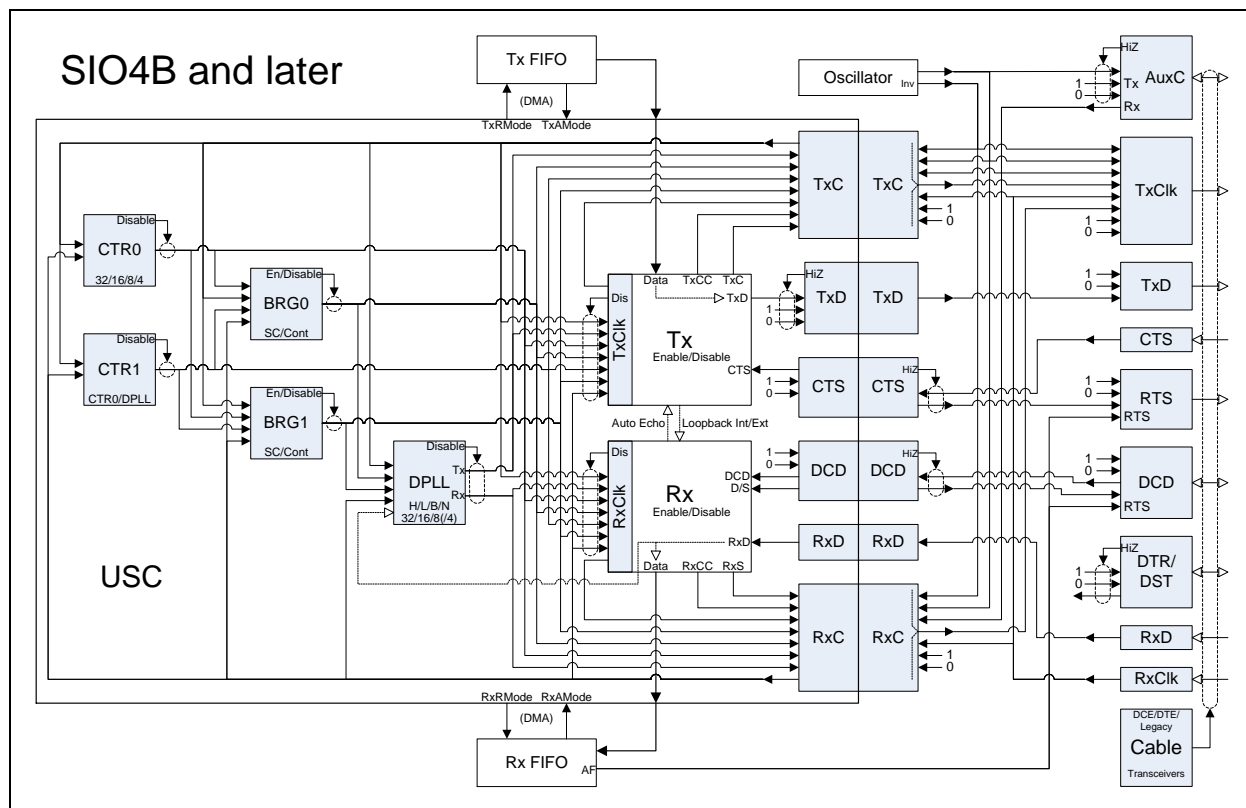


Figure 2 A functional illustration of an SIO4B or later model board.

5.2. Getting Started

To configure the SIO4 for HDLC operation meeting specific requirements, the recommended starting point is a local, customizable copy of the `hdlcc2c` sample application included with the driver. This application is designed to transfer bulk data between an SIO4 transmitter and an SIO4 receiver. The transmitter and receiver can be from two channels on different boards, two channels on the same board, or by loopback mode using the transmitter and receiver from the very same channel. There are two loopback configurations. Internal Loopback performs the transfer by routing signals totally onboard the SIO4 without driving the cable transceivers. External Loopback performs the transfer by routing the signals through the cable transceivers.

NOTE: When using Loopback operation, it is recommended that cabling and any remote equipment be disconnected from the SIO4. This is because External Loopback is the default when Internal Loopback is not supported by firmware.

5.2.1. Cable Validation

The first step in deriving a customized configuration is to verify cabling. This should be done using the application's default settings. It is recommended that one channel be used for loopback testing and that two other channels on the

same board be connected by the cabling to be tested. A script with the below commands is a convenient means of repeating these tests until cabling has been verified successfully.

```
./hdlcc2c 0 0 -i
./hdlcc2c 0 0 -e
./hdlcc2c 1 2
```

5.2.2. Customizing the Configuration

The second step in deriving a customized configuration is to methodically modify the application code, one parameter at a time, until all necessary parameter changes have been accommodated. That is, choose a parameter from the documentation that must be changed from its default, modify the application so the required setting can be specified from the command line, then test the resulting changes. A script with the below commands is a convenient means of repeating these tests. In this example the “-X” represents the new command line argument for the parameter being altered from its default. This script tests all three cabling setups both without the change and with the change. This is done to verify that the default operation remains functional after the code modifications. In some cases, the script may need to be expanded to test each parameter addition to ensure prior changes remain functional. It is suggested that parameter changes be accommodated one at a time to ease the development and testing process.

```
./hdlcc2c 0 0 -i
./hdlcc2c 0 0 -e
./hdlcc2c 1 2 -i
./hdlcc2c 0 0 -i -X
./hdlcc2c 0 0 -e -X
./hdlcc2c 1 2 -X
```

NOTE: At times modifications for a parameter may need to be implemented on the transmitter or receiver first in order to facilitate validation. At other times the transmitter and receiver may temporarily be configured differently in order to verify that a change is implemented properly.

NOTE: It is best to initially test parameter additions separately, one at a time. Where there are parameter interactions, testing parameter combinations should be completed before moving on.

The general sequence for addition of a new parameter modification is as follows.

1. Add a field representing the parameter to the `args_t` structure defined in `main.h`.
2. Add command line support for the new parameter by updating the `_parse_args()` function at the top of `main.c`. At minimum, the `list[]` table must be updated to associate assignment of a setting with a command line argument. This may also mean assigning a default to the new field following the `memset()` function call.

NOTE: One may have to review multiple sample applications to get a feel for how to add specific command line argument types to the argument table.

3. Update the `_setup_apply()` function at the top of `setup.c` to apply the value from the new field to the `sio4_hdlc_t` structure prior to calling `sio4_hdlc_set()`.
4. Now update the script steps given above for the code changes just implemented. Continue adding support for other required parameter changes when testing is complete. Update the above script for each addition.

5.3. Debugging Aids

The SIO4 driver archive includes the following debugging aids appropriate for use with the HDLC Protocol Library.

5.3.1. Device Identification

When communicating with technical support complete device identification is virtually always necessary. The *id* example application is provided for this specific purpose. This is a text only console application. The output can be piped to a file, which can then be emailed to GSC technical support when requested. Locate the application as follows.

Description	File	Location	OS
Application	id	.../id/	Linux
	id.rta	...\\id\\	INtime

5.3.2. sio4_hdlc_show()

The function `sio4_hdlc_show()` (section 4.1.10, page 20) is part of the protocol library interface. The purpose of the function is to produce a human readable report of all fields included in the `sio4_hdlc_t` structure (section 4.2.1, page 26) passed in as a function argument. The function is best used to report the structure's content before it is passed to `sio4_hdlc_set()` (section 4.1.9, page 19) or after it is passed to `sio4_hdlc_get()` (section 4.1.2, page 12). The output can be used with Figure 2 to help visualize the channel configuration reflected by the structure content. When used in conjunction with `sio4_hdlc_set()`, the `sio4_hdlc_show()` output indicates the state that `sio4_hdlc_set()` is expected to produce. When used in conjunction with `sio4_hdlc_get()`, the `sio4_hdlc_show()` output indicates the channel's current state. This may be beneficial after calling `sio4_hdlc_set()` in order to verify the results achieved. The pair of calls may also be used before or after read or write requests in order to help explain the results of individual transfer requests.

5.3.3. Detailed Register Dump

The utility function `sio4_reg_list()` is included in the sio4 utility library. The purpose of the function is to report the current content of registers for the referenced serial channel. The arguments control the set of registers included in the output and the detail with which the register content is reported. This function can be called at any time to report the device state, but it is most often called after completing board setup, or just before or after read or write requests in order to help explain the results of individual transfer requests.

Prototype

```
int sio4_reg_list(int fd, int gsc, int gsc_detail,
                 int usc, int usc_detail);
```

Argument	Description
fd	This is a file descriptor obtained from <code>sio4_hdlc_open()</code> (section 4.1.6, page 15).
gsc	If non-zero, then the output will include a dump of all GSC_SIO4_XXX registers. Refer to <code>sio4.h</code> for a complete list of these registers.
gsc_detail	If non-zero, then the dump of the above registers will include detailed information about all register fields, including the field value and the meaning of the value.
usc	If non-zero, then the output will include a dump of all GSC_USC_XXX registers. Refer to <code>sio4.h</code> for a complete list of these registers.
usc_detail	If non-zero, then the dump of the above registers will include detailed information about all register fields, including the field value and the meaning of the value.

Return Value	Description
>= 0	This is the number of errors encountered during execution of the function.

5.3.4. Status Return Values

The HDLC Protocol Library, the SIO4 API Library and the SIO4 device driver all report the results for each of the various interface services. The table below lists the most common error status values reported to an application.

NOTE: When an error status is returned by the HDLC Init Data, Get, Set, Show and Tx Frame functions, the `err` argument to these same functions provides a string that identifies the offending argument or structure field.

Value	<code>errno.h</code> Macro	Description
0	None	The operation was completed successfully.
-16	<code>-EBUSY</code>	An open request failed due to a conflicting <code>share</code> argument. This can happen if an Exclusive open request is made when some application already has access to the same device. This will occur when the existing access is either Shared or Exclusive. This can also happen if a Shared request is made when the existing access is Exclusive.
-22	<code>-EINVAL</code>	A function argument or referenced structure field value is invalid.
-71	<code>-EPROTO</code>	The HDLC Protocol Library has not been initialized. Applications must call <code>sio4_hdlc_init()</code> before making any other Library call.
-77	<code>-EBADF</code>	The file descriptor argument was not recognized. This indicates that the file descriptor is invalid, was not obtained by the <code>sio4_hdlc_open()</code> function, or access to the referenced device has already been closed.
-93	<code>-EPROTONOSUPPORT</code>	This indicates that the SIO4 device doesn't support the operation requested. This occurs when any of the functions listed in the above note is called on an SIO4 that is not based on the Zilog Z16C30 DUART. The board is either a -SYNC mode or a model with custom firmware.

5.4. Tx Abort (Transmitter Abort Sequence)

The USC can induce an Abort condition onto the Tx Data output under two conditions. It is done either on demand by calling `sio4_hdlc_tx_abort()` (see section 4.1.11, page 20) or automatically if it runs out of data while transmitting a frame. When done by calling the above function the Abort signal is the shorter seven-bit version. When done automatically, the USC can be configured for either the shorter seven-bit version or the longer 15-bit version. The length selection is made via the USC's Tx Underrun feature configuration (see `sio4_hdlc_t.tx.underrun`, section 4.2.1, page 26).

NOTE: If an Abort sequence is requested near the end of an HDLC frame, then the frame may be terminated prematurely. When this occurs, the data at the tail end of the frame may not be transmitted. Additionally, the CRC, if enabled, may not be transmitted. Correspondingly, the transmitter may not generate the CRC Sent and EOF Sent interrupts.

5.5. Tx Preamble (Transmitter Frame Preamble)

The USC supports generation of a Preamble before transmitting each frame (see `sio4_hdlc_t.tx.preamble`, section 4.2.1, page 26). By default the Preamble is disabled. If enabled, the default pattern is a single Flag sequence. The Preamble length is configurable as eight, 16, 32 or 64-bits. If the `flag` field is non-zero and the `pattern` field is the "1" pattern option, then the Preamble pattern is the Flag sequence. Otherwise the pattern is per the `pattern` field.

5.6. Tx CRC (Transmitter Frame Check Sequence)

The USC supports generation of a CRC following transmission of the application provided frame data (see `sio4_hdlc_t.tx.crc`, section 4.2.1.3, page 34). By default the CRC is enabled and configured to append a 16-bit CCITT CRC, with a "1" preset, at the end of each frame. The `enable` field, if non-zero configures the

transmitter to calculate the CRC as the frame is being transmitted. The `on_end` field, if non-zero, configures the transmitter to send the CRC before closing the frame. If this field is zero, then the CRC is calculated but not sent.

NOTE: Please refer to the *Zilog Z16C30 Databook* before changing the CRC type or preset.

NOTE: The CRC is appended to the application provided data sent as part of the frame. And as the USC receiver is limited to tracking frame sizes from one to 65,535 bytes (1 to 0xFFFF), inclusion of the CRC correspondingly reduces the user data limit in a frame by the size of the CRC.

5.7. Rx CRC (Receiver Frame Check Sequence)

The USC supports CRC checking of received frames (see `sio4_hdlc_t.rx.crc`, section 4.2.1.4, page 38). By default the CRC is enabled and configured to use the 16-bit CCITT CRC, with a “1” preset. The `enable` field, if non-zero configures the receiver to calculate the CRC as the frame is being received.

NOTE: Please refer to the *Zilog Z16C30 Databook* before changing the CRC type or preset.

NOTE: The USC does not strip the received CRC from the received data stream. And as the USC receiver is limited to tracking frame sizes from one to 65,535 bytes (1 to 0xFFFF), inclusion of the CRC correspondingly reduces the user data limit in a frame by the size of the CRC.

5.8. Basic Transmission Requirements

The HDLC Protocol Library accommodates various application frame transmission requirements, as described below. Satisfying different requirements is achieved by varying the arguments passed to the `sio4_hdlc_tx_frame()` service (section 4.1.14, page 22). These examples are given simply to illustrate how to the Library can be used to support some common requirements.

5.8.1. Send Data as a Single Complete Frame

The library can conditionally guarantee that a Tx Frame is sent out in its entirety as a single complete frame. The conditions that apply are that the frame content must fit within the SIO4's Tx FIFO and that the frame be configured to have its content preloaded into the Tx FIFO before initiating frame transmission. If the frame content exceeds the Tx FIFO size, preloading can still be performed. In these cases, after preloading, the frame portion that remains is sent to the FIFO after frame transmission is initiated. The table below illustrates the settings that will request the preloading option.

<code>sio4_hdlc_tx_frame_t</code> Field	Value
<code>preload</code>	1
<code>size</code>	As required.
<code>src</code>	As required.
<code>last</code>	As required.
<code>wait</code>	0
<code>sent</code>	0
<code>status</code>	0

Preloading helps minimize the likelihood that a Tx Frame ends prematurely. Options that can increase the likelihood of the data being sent as a single complete frame are: increase the Tx FIFO size, reduce the frame data size, enable maximum Preamble size, reduce the transmission bit rate and increase the execution priority of the thread sending the frame relative to other application and/or system activities.

NOTE: The preload option can reduce overall transmission throughput because the library waits for the Tx FIFO to empty before proceeding. If this is an issue for an application, then tests should be performed to determine if preloading is necessary.

NOTE: The prefill operation is performed only after the data for any prior frames has left the Tx FIFO. The condition is checked by polling at an interval of one system timer tick.

NOTE: If a data set is not transmitted via a single complete frame, then upon return the `status` field will include the flag `SIO4_HDLC_TX_FRAME_STATUS_SHORT`. This error condition pauses USC transmission necessitating a recovery operation.

NOTE: The driver inserts four bytes of overhead into the Tx FIFO at the start of each HDLC Tx Frame, which reduces the available Tx FIFO space by four bytes. This also reduces the preload capacity by four bytes.

5.8.2. Tx Short Frame Status and Preload

The Short Frame status indicates that software was not able to provide frame data fast enough to keep data in the Tx FIFO before it is needed by the USC. The higher the bit rate the more likely this is to occur. The best way to address this is set the `preload` field to one. If Tx Frames are sent back-to-back and are of sufficient size, then this may be needed only for the first frame following a notable break. If frames are relatively small, then this may be needed for every frame. Experimentation may be required to determine the best approach for each usage.

The library can conditionally guarantee that a Tx Frame is sent out in its entirety as a single complete frame. The conditions that apply are that the frame content must fit within the SIO4's Tx FIFO and that the frame be configured to have its content preloaded into the Tx FIFO before initiating frame transmission. If the frame content exceeds the Tx FIFO size, preloading can still be performed. In these cases, after preloading, the frame portion that remains is sent to the FIFO after frame transmission is initiated. The table below illustrates the settings that will request the preloading option.

5.8.3. Complete Frame Transmission Before Returning

When an application needs confirmation that a frame has been transmitted completely it can make that request when the Tx Frame service is called. The confirmation is accomplished by telling the service the condition that the service is to wait for after all data has been written to the Tx FIFO. The table below illustrates the settings that will request transmission completion before returning.

<code>sio4_hdlc_tx_frame_t</code> Field	Value
<code>single</code>	0
<code>size</code>	As required.
<code>src</code>	As required.
<code>last</code>	As required.
<code>wait</code>	<code>SIO4_HDLC_TX_FRAME_STATUS_EOF</code>
<code>sent</code>	0
<code>status</code>	0

NOTE: The `wait` field can specify any number of wait conditions. The service will cease waiting upon detection of the first condition, or when the I/O timeout expires, whichever occurs first.

NOTE: The waiting operation is performed by polling at an interval of one system timer tick.

NOTE: For additional information refer to the Tx Abort service (section 4.1.11, page 20).

5.8.4. Transmit Frames ASAP

The Tx Frame service will send data ASAP if all waiting conditions are disabled. This includes the `single` field and the `wait` field. The table below illustrates the settings that will request transmission ASAP.

<code>sio4_hdlc_tx_frame_t</code> Field	Value
<code>single</code>	0
<code>size</code>	As required.
<code>src</code>	As required.
<code>last</code>	As required.
<code>wait</code>	0
<code>sent</code>	0
<code>status</code>	0

5.9. Clocking Configurations

The function `sio4_hdlc_init_data()` initializes the USC clocking section of the `sio4_hdlc_t` structure based on the content of the `sio4_hdlc_init_t` structure. The basic configuration options are shown below. In all cases, unused USC clocking components are disabled. The following illustrations are for SIO4B or later model boards. The figures are also representative of SIO4A model boards, except that the SIO4A has lower capabilities for routing signals between the USC and the cable interface. The figures are also somewhat representative of the basic SIO4 model boards, except that the basic SIO4 boards use jumpers for routing signals between the USC and the cable interface. (The basic SIO4 model boards are limited to the legacy mode cable configuration feature.) The SIO4 USC's each having two clocking pins usable as input or output clocks; `TxC` and `RxC`. The cable clocking signals include the Cable Tx Clock (`TxClk`) and the Cable Rx Clock (`RxClk`). There is also a Cable Auxiliary Clock (`AuxC`), which can be an output (`TxAuxC`) or an input (`RxAuxC`), but that signal doesn't affect the limitations of having only two USC pins usable for clocking purposes. The selection of which cable clock signals are used, `TxClk` and/or `RxClk`, affects the flexibility and limitations on how the USC clock pins are used (`TxC` and/or `RxC`, respectively). The SIO4 and the USC have extensive clocking capabilities. Many more options are possible besides the four basic configurations given below. When calling `sio4_hdlc_init_data()` however, these four configurations are the only ones evaluated. If an alternate setup is required, then the settings required should be applied to the `sio4_hdlc_t` structure between the calls to `sio4_hdlc_init_data()` and `sio4_hdlc_set()`.

5.9.1. Four Signal Configuration: Cable Tx Clock is Used, Cable Rx Clock is Used

NOTE: This is a preferred configuration as a clock is provided with each data signal. The clocks are therefore perfectly timed with the data being transferred.

This is a four signal configuration using the Cable Tx Data signal (TxD), the Cable Tx Clock signal (TxClk), the Cable Rx Data signal (RxD) and the Cable Rx Clock signal (RxClk). As the Cable Rx Clock signal (RxClk) is used, the USC Rx Clock pin (RxC) is dedicated to that purpose. At the same time, the Cable Tx Clock signal (TxClk) is used and the USC Tx Clock pin (TxC) is required as an input. To satisfy these requirements, the on-board oscillator is programmed to the Tx bit rate and configured as the source for both Cable Tx Clock (TxClk) signal and the USC Tx Clock pin (TxC) . **Therefore, the `osc_prog` field must be set to the value in the `tx_bit_rate` field.** This signal routing is illustrated in Figure 3. The heavy dotted lines with the yellow background are the hard coded routing selections for this configuration. The code extract below demonstrates the minimum coding for this example configuration. Error checking is omitted for brevity. The resulting bit rates are reported in the `sio4_hdlc_t` structure and should be examined for suitability.

```
const char*      err      = NULL;
sio4_hdlc_t      hdlc;
sio4_hdlc_init_t init;

init.tx_bit_rate    = 150000;
init.rx_bit_rate    = 150000;
init.cbl_txc        = SIO4_HDLC_CBL_TXC_USED;
init.cbl_rxc        = SIO4_HDLC_CBL_RXC_USED;
init.osc_prog       = 150000; // tx_bit_rate
sio4_hdlc_init_data(fd, &init, &hdlc, &err);
```

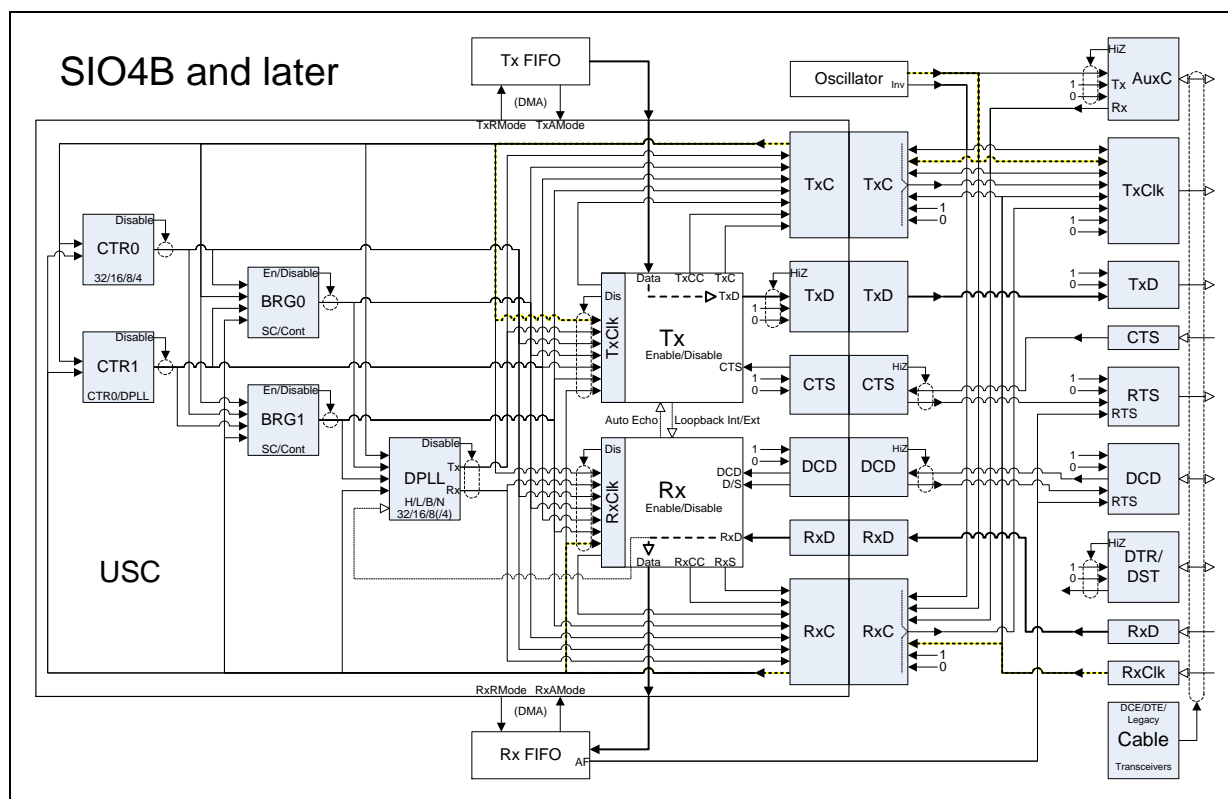


Figure 3 The clock routing produced when the Cable Tx Clock is used and the Cable Rx Clock is used.

5.9.2. Three Signal Configuration #1: Cable Tx Clock is Unused, Cable Rx Clock is Used

NOTE: This may not be a preferred configuration as the remote device will have to construct a clock from transitions in the Tx Data signal. While this is common, the remote device may not be able to construct a clock that exactly matches the original transmit clock.

This is a three signal configuration using the Cable Tx Data signal (TxD), the Cable Rx Data signal (RxD) and the Cable Rx Clock signal (RxClk). As the Cable Rx Clock signal (RxClk) is used, the USC Rx Clock pin (RxC) is dedicated to that purpose. The USC Tx Clock pin (TxC) is therefore dedicated as the input from the on-board programmable oscillator. The USC Tx Clock (TxClk) is derived from the on-board programmable oscillator. This signal routing is illustrated in Figure 4. The heavy dotted lines with the yellow background are hard coded routing selections. The heavy dashed lines with the yellow background are the possible routing selections evaluated to produce the specified Tx bit rate. The code extract below demonstrates the minimum coding for this example configuration. Error checking is omitted for brevity. **For the best possible bit rate matching, the `osc_prog` field must be set an even multiple of the value in the `tx_bit_rate` field, but not higher than 20,000,000. Higher is better, though.** The resulting bit rates are reported in the `sio4_hdlc_t` structure and should be examined for suitability.

```
const char*      err      = NULL;
sio4_hdlc_t      hdlc;
sio4_hdlc_init_t  init;

init.tx_bit_rate    = 150000;
init.rx_bit_rate    = 150000;
init.cbl_txc        = SIO4_HDL_CBL_TXC_UNUSED;
init.cbl_rxc        = SIO4_HDL_CBL_RXC_USED;
init.osc_prog       = 19950000; // 133 * tx_bit_rate
sio4_hdlc_init_data(fd, &init, &hdlc, &err);
```

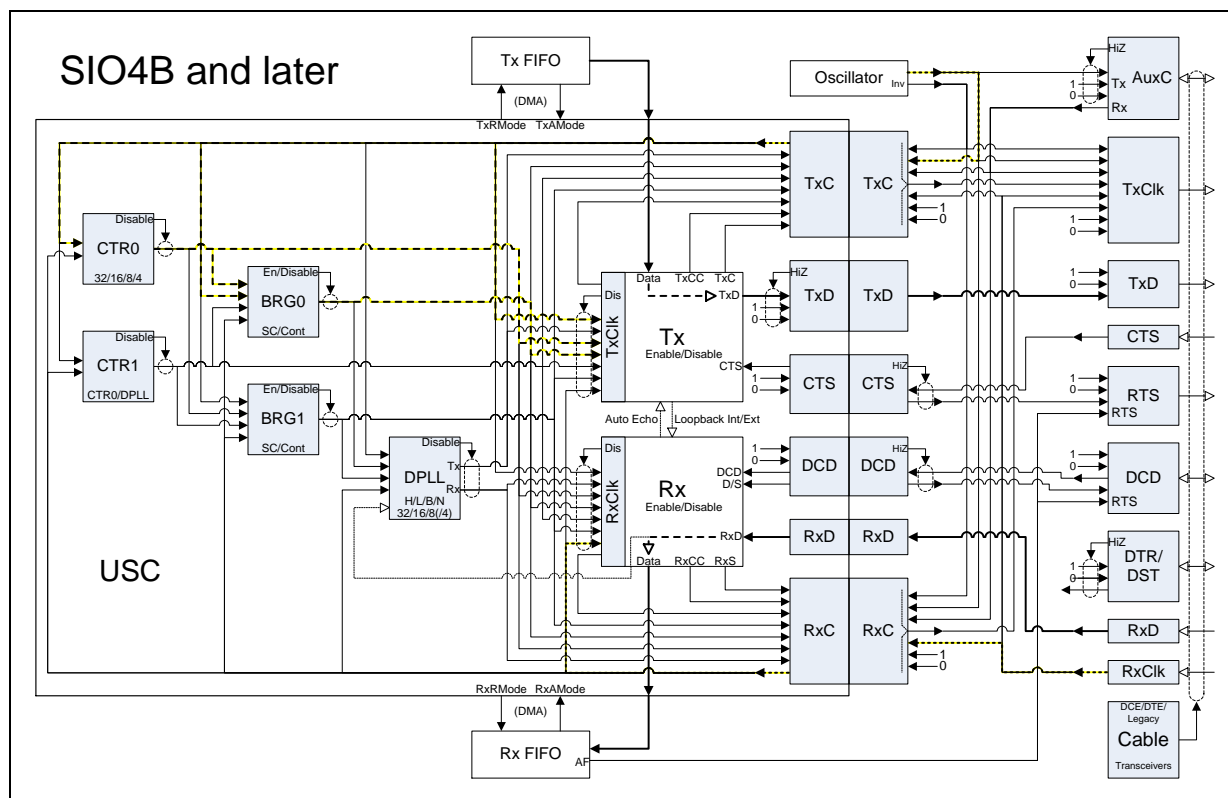


Figure 4 The clock routing produced when the Cable Tx Clock is unused and the Cable Rx Clock is used.

5.9.3. Three Signal Configuration #2: Cable Tx Clock is Used, Cable Rx Clock is Unused

NOTE: This may not be a preferred configuration as the receiver will have to construct a clock from transitions in the Rx Data signal. While this is common, the receiver may not be able to construct a clock that exactly matches the original transmit clock.

This is a three signal configuration using the Cable Tx Data signal (TxD), the Cable Tx Clock signal (TxClk) and Cable Rx Data signal (RxD). As the Cable Tx Clock signal (TxClk) is used, the USC Tx Clock pin (TxC) is dedicated to that purpose. The USC Rx Clock pin (RxC) is therefore dedicated as the input from the on-board programmable oscillator. The USC Rx Clock (RxClk) is extracted from the Cable Rx Data signal (RxD) by the DPLL. The USC Tx Clock (TxClk) is derived from the onboard oscillator. This is illustrated in Figure 5. The heavy dotted lines with the yellow background are hard coded routing selections. The heavy dashed lines with the yellow background are the possible routing selections. The code extract below demonstrates the minimum coding for this example configuration. **For the best possible bit rate matching, the `osc_prog` field must be set to eight, 16 or 32 times the value in the `rx_bit_rate` field, but not higher than 20,000,000. Higher is better, though. It should also be an even multiple of the `tx_bit_rate` field, should the bit rates differ.** The resulting bit rates are reported in the `sio4_hdlc_t` structure and should be examined for suitability.

```
const char*      err      = NULL;
sio4_hdlc_t      hdlc;
sio4_hdlc_init_t init;

init.tx_bit_rate = 150000;
init.rx_bit_rate = 150000;
init.cbl_txc     = SIO4_HDLC_CBL_TXC_USED;
init.cbl_rxc     = SIO4_HDLC_CBL_RXC_UNUSED;
init.osc_prog    = 19200000; // 32 * rx_bit_rate * 4
sio4_hdlc_init_data(fd, &init, &hdlc, &err);
```

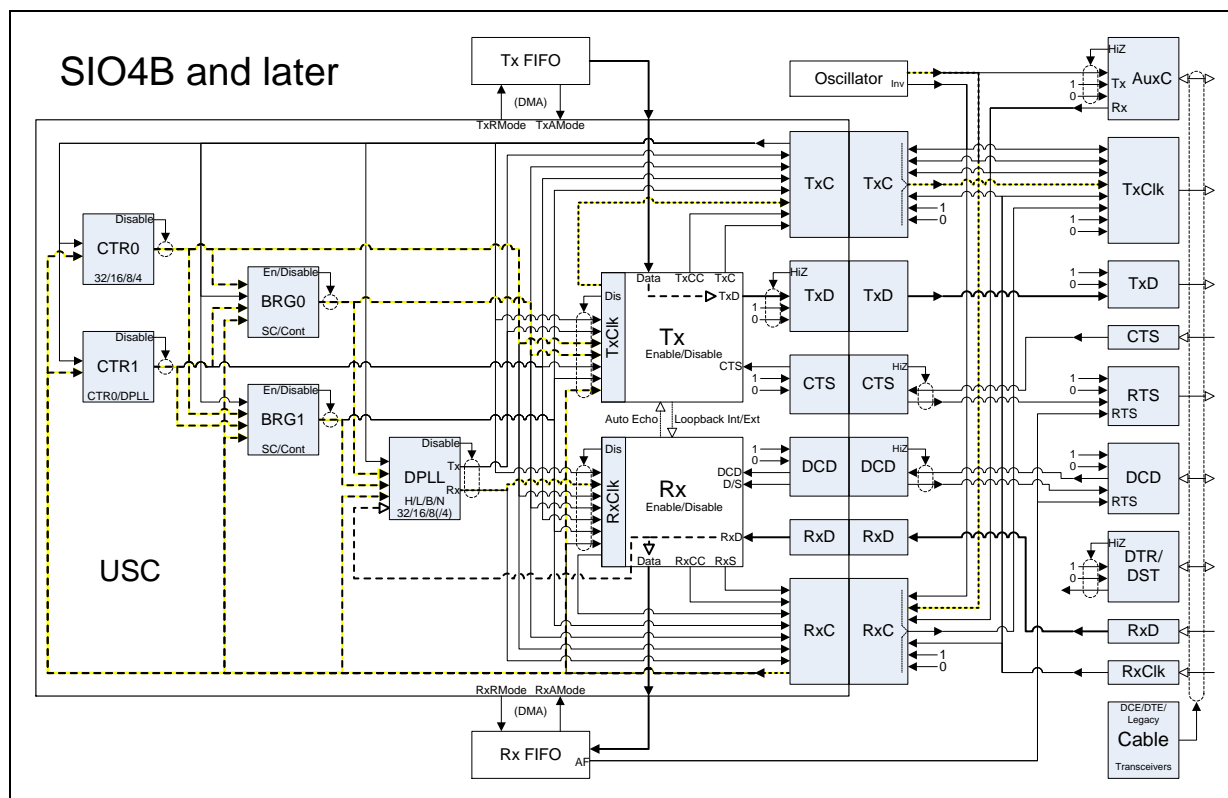


Figure 5 The clock routing produced when the Cable Tx Clock is used and the Cable Rx Clock is unused.

5.9.4. Two Signal Configuration: Cable Tx Clock is Unused, Cable Rx Clock is Unused

NOTE: This may not be a preferred configuration as the local and remote receivers will have to construct clocks from transitions in their Rx Data signals. While this is common, the receivers may not be able to construct clocks that exactly match the original transmit clocks.

This is a typical two signal configuration using only the Cable Tx Data signal (TxD) and the Cable Rx Data signal (RxD). As the cable clock signals are unused, the USC clocking pins (TxC and RxC) can be used as needed, giving maximum flexibility. The USC Rx Clock (RxClk) is extracted from the Cable Rx Data signal (RxD) by the DPLL using the onboard oscillator. The USC Tx Clock (TxClk) is derived from the onboard oscillator. This signal routing is illustrated in Figure 6. The heavy dotted lines with the yellow background are hard coded routing selections. The heavy dashed lines with the yellow background are the possible routing selections evaluated to produce the specified bit rates. The code extract below demonstrates the minimum coding for this example configuration. Error checking is omitted for brevity. **For the best possible bit rate matching, the `osc_prog` field must be set to eight, 16 or 32 times the value in the `rx_bit_rate` field, but not higher than 20,000,000. Higher is better, though. It should also be an even multiple of the `tx_bit_rate` field, should the bit rates differ.** The resulting bit rates are reported in the `sio4_hdlc_t` structure and should be examined for suitability.

```
const char*      err      = NULL;
sio4_hdlc_t      hdlc;
sio4_hdlc_init_t init;

init.tx_bit_rate = 150000;
init.rx_bit_rate = 150000;
init.cbl_txc     = SIO4_HDLC_CBL_TXC_UNUSED;
init.cbl_rxc     = SIO4_HDLC_CBL_RXC_UNUSED;
init.osc_prog    = 19200000; // 32 * rx_bit_rate * 4
sio4_hdlc_init_data(fd, &init, &hdlc, &err);
```

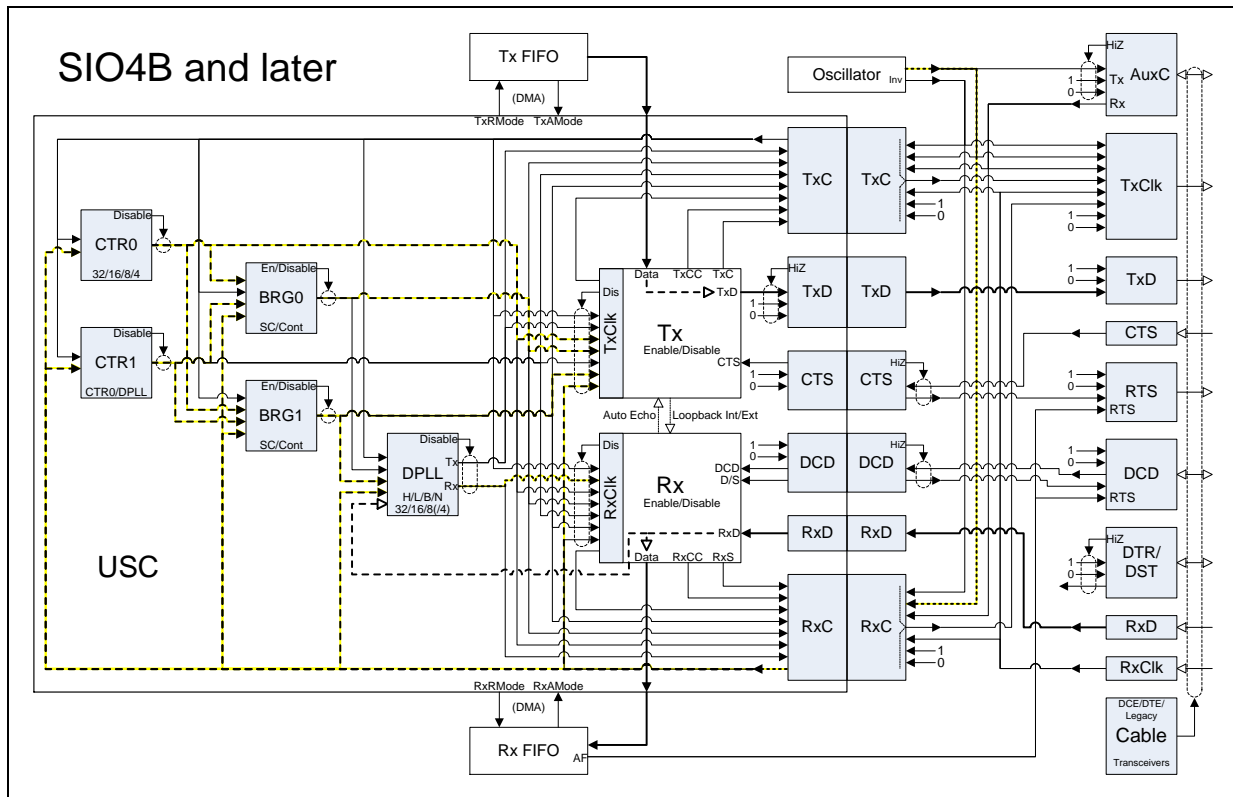


Figure 6 The clock routing produced when the signals Cable Tx Clock and Cable Rx Clock are unused.

5.10. Cable Configuration Modes

The SIO4 supports two cable interfacing modes; DCE/DTE mode and Legacy mode. Older boards support only Legacy mode. More recent boards support only DCE/DTE mode. Intermediate boards support both modes. When both are available selection of the active mode is governed by enabling or disabling the transceivers. This is done through the `sio4_hdlc_t.cable.enable` field, which is described under section 4.2.1.2 beginning on page 30.

5.10.1. DCE/DTE Mode

The DCE/DTE Mode controls cable signaling according to the DCE or DTE selection, as described in the board user manual. When available in firmware this mode is enabled by enabling the cable transceivers (see paragraph above). The HDLC Protocol Library passes all DCE/DTE mode settings to the driver unconditionally. DCE/DTE mode settings received by the driver are applied only if this mode is supported by the board. The driver otherwise ignores these settings.

5.10.2. Legacy Mode

The Legacy Mode of operation controls signal routing for the cable interface, according to the Upper and Lower settings, and the USC, as described in the board user manual. When available in firmware this is the default mode of operation. When selectable this mode is activated by disabling the cable transceivers (see paragraph above). The HDLC Protocol Library's `sio4_hdlc_set()` function (section 4.1.9, page 19) applies Legacy mode settings only if the Legacy mode will be active when the function exits. The library otherwise ignores the Legacy settings. All other library functions process their settings unconditionally. Legacy mode settings received by the driver are applied only if this mode is supported by the board. The driver otherwise ignores these settings.

5.11. Error and Status Detection

The serial controller used on the SIO4 incorporates the ability to detect a number of error and other conditions for both the transmit and the receive data streams.

5.11.1. Interrupt Events

The most efficient means of detecting the various conditions, especially errors, is by use of interrupts. The basic steps for this are to enable the interrupts of interest then have a thread wait for a corresponding interrupt event. (See the Interrupt and the Wait Event services in the driver reference manual.) This is illustrated in the following code fragments.

Thread A	Thread B
<pre> For (;;) { ... read SIO4 data if (error recorded) { Error exists in Read buffer, Or SIO4 Rx FIFO Resync data stream. } else { Read buffer is error free. } ... </pre>	<pre> For (;;) { ... Enable desired interrupts. Wait for an interrupt. if (error interrupt occurred) { Record the error. } ... } </pre>

}	
---	--

5.11.2. Rx Status Word

The SIO4 can also provide status in the Rx data stream on a per byte basis. This is done by enabling the Rx Status Word feature (`sio4_hdlc_t.rx.status_word`, section 4.2.1.4, page 38). When enabled, the SIO4 places the lower eight bits of the USC's Receive Command/Status Register in the Rx FIFO immediately after the data itself. This allows an application to identify the precise location in the data stream where some Rx related conditions occur. The downside of this is that it doubles the volume of data going through the Rx FIFO and effectively reduces its size by 1/2. Refer to the *Z16C30 Data Handbook* for information on the USC's Receive Command/Status Register.

Document History

Revision	Description
March 19, 2024	Updated release date. Added comment about use of the static libraries. Removed the Cable Protocol Disable option as it is unsupported by hardware.
November 16, 2023	Updated release date.
June 15, 2023	Updated release date. Minor editorial updates.
December 13, 2022	Updated release date. Minor editorial updates.
June 2, 2022	Updated release date. Updated the description of the Tx Idle Line conditions.
February 8, 2022	Updated the release date. Expanded the description of the Cable Tx and Rx Clock signal options. Removed “PRELIMINARY” status. Changed the description of the Tx Size Limit field. Added clarification of the Tx Frame wait field options. Updated notes for open requests on devices which do not support HDLC. Reorganized the Operating Information section (section 5, page 50). Added a Getting Started section (section 5.2, page 50). Added notes about the driver’s internal RCC FIFO. Added notes for the Tx Frame and Rx Frame status fields. Added information on common error values returned by the Library. Minor editorial modifications. Corrected the description of the <code>sio4_hdlc_tx_wait()</code> function. Added the argument “file” to the Show service .
August 9, 2021	Updated the release date. Removed “PRELIMINARY” status. Updated information on the Tx Frame Abort call. Updated the I/O timeout field descriptions. Updated the descriptions of the I/O return values. Updated the content and description of the <code>sio4_hdlc_rx_frame_t</code> data structure. Updated the description of the <code>sio4_hdlc_tx_frame_t</code> data structure. Added notes about Rx Frame and Rx Flush requests being serialized. Updated the notes about Tx Frame, Tx Flush and Tx Status requests being serialized. Added statements about the Tx Frame and Rx Frame services being blocking calls.
May 5, 2021	Updated the release date.
February 26, 2021	Updated the release date.
February 18, 2021	Updated information on the Tx and Rx DMA Threshold settings.
February 1, 2021	Updated the release date. Updated the description of the <code>sio4_hdlc_init()</code> function. Updated the description of the <code>sio4_hdlc_init_data()</code> function.
October 12, 2020	Updated the release date. Minor editorial corrections. The minimum frame data size is two bytes. The maximum frame data size is 0xFFFF minus the size of the CRC. Revised the <code>sio4_hdlc_tx_frame_t.wait</code> options.
July 30, 2019	Updated the release date. Added information on state of the preliminary release.
March 24, 2019	Updated the release date.
March 15, 2019	Added the <code>sio4_hdlc_t.rx.size_limit</code> field. Remove references to the low level services. Removed all references to the Library interface being high level services. Added <code>sio4_hdlc_tx_frame_status()</code> .
October 18, 2018	Updated the default PIO threshold values based on testing. Updated the inside cover page. Added Tx and Rx I/O DMA Threshold fields.
October 31, 2017	Added information on the Cable Configuration Modes. Updated some legacy setting information.
October 17, 2017	Removed library versioning. Added information on opening device index -1. Added section on library interface files. Updated return status information for high level services. Added support for device index -1. Numerous editorial changes throughout.
December 7, 2016	Updated the operating information section. Made various miscellaneous updates. Updated information on using loopback mode.
September 16, 2016	Updated to version 0.11. Updated the <code>sio4_hdlc_init_t</code> structure. Updated the text of the Rx Address Control macros. Changed the defaults for several settings.
April 13, 2016	Updated to version 0.10. Updated the clocking configuration and event detection information.
September 7, 2015	Updated to version 0.9.
December 9, 2014	Updated the current release date.

December 4, 2014	Updated to version 0.8. Added information on DCD configuration for both the cable and the USC. Moved the data structure section into the previous section. Added information on error and status detection.
May 17, 2014	Updated to version 0.7. The two clocking configuration options that required the DPLL were replaced by a single option that derived the Rx Clock from the DPLL and the Tx Clock from the onboard programmable oscillator.
April 16, 2014	Updated to version 0.6.
October 22, 2013	Updated to version 0.5.
October 15, 2013	Updated to version 0.4.
August 27, 2013	Initial release, version 0.2. This is for the 2.x series driver.