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: September 7, 2015
Library Release Version: 0.9

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

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 material, 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 to ECO control, **General Standards Corporation** assumes no responsibility for any errors that may exist in this document. No commitment is made to update or keep current the information contained in this document.

General Standards Corporation does not assume any liability arising out of the application or use of any 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 product to improve reliability, performance, function, or design.

The Purchaser of this software may use or modify in source form the subject software, but not to re-market or distribute it to outside agencies or separate internal company divisions. The software, however, may be embedded in the Purchaser's distributed software. In the event the Purchaser's customers require the software source code, then they would have to purchase their own copy of the software.

General Standards Corporation makes no warranty of any kind with regard to this software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and makes this software available solely on an "as-is" basis. **General Standards Corporation** reserves the right to make changes in this software without reservation and without notification to its users.

The information in this document is subject to change without notice. This document may be copied or reproduced provided it is in support of products from **General Standards Corporation**. For any other use, no part of this document may be copied or reproduced in any form or by any means without prior written consent of **General Standards Corporation.**

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

Table of Figures

1. Introduction

This document provides information on the HDLC Protocol Library, which is a library designed to facilitate use of the HDLC serial protocol with an SIO4 or SIO8. This library version is designed to work with the 2.x series driver released on the same day as this version of the library.

1.1. Purpose

The purpose of this document is twofold. First, it is intended to give a basic description the HDLC framing architecture. Second, it is intended to give a complete description of the HDLC Protocol Library interface.

1.2. Acronyms

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

Acronyms	Description
DMA	Direct Memory Access
DPLL	Digital Phase Lock Loop
GSC	General Standards Corporation
HDLC	High-level Data Link Control
PCI	Peripheral Component Interconnect
PMC	PCI Mezzanine Card
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 the user space with user mode privileges.
Driver	Driver means the executable programming providing the direct access to the SIO4 hardware.
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.

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 device driver. 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. Applications are free to use the library interface and the driver interface at will. The only requirements are that applications must use the library's `sio4_hdlc_lib_init()`, `sio4_hdlc_open()` and `sio4_hdlc_close()` functions.

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 the SIO4 and remote devices. The SIO4 board includes two DMA controllers and comes with a maximum of 256K Bytes of FIFO storage, which is 32K per channel per direction (32K * 2 * 4). The FIFO configuration can vary greatly from one SIO4 version to another (i.e. 32K * 2 * 4 to 512 * 2 * 2 to none at all). The SIO4 can be configured with a number of different cable transceivers. The transceivers can be fixed as RS232 or

RS422, or they can be changed programmatically to a number of different options. The Zilog version of the SIO4 supports a number of different serial protocols by use of a pair of dual Universal Serial Controllers (the Zilog Z16C30). This includes Asynchronous, Monosync, Bisync, HDLC and a few others.

1.6. 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. The normal Abort is a zero followed by seven ones. The Extended Abort is a zero followed by 15 ones. When sending out other than these special sequences, the transmitter performs bit stuffing by converting any 011111 sequence into 0111101. The receiver performs the reverse operation by converting the sequence 0111101 to 011111. In this way, the HDLC protocol places no restriction on the content of the serial data.

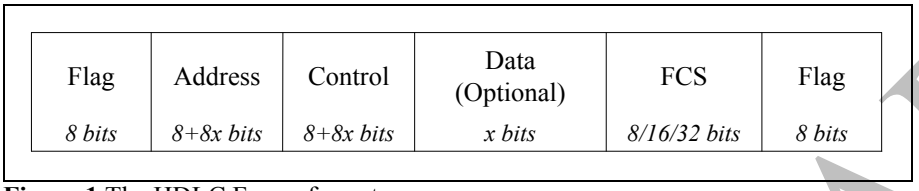

Figure 1 The HDLC Frame format.

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 eight, 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. Functions

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 the library header file 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 functions, structures, and macros.

3.1. High Level Functions

The high level functions are described below.

3.1.1. sio4_hdlc_lib_init()

This function is required in order to prepare the HDLC Protocol Library for operation. This must be the first call to the library.

Prototype

```
int sio4_hdlc_lib_init(void);
```

Argument	Description
None	The function has no arguments.

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

3.1.2. sio4_hdlc_version()

This function retrieves version and build information about the library.

Prototype

```
int sio4_hdlc_version(const char** version, const char** built);
```

Argument	Description
`fd`	This is a file descriptor obtained by a call to `sio4_hdlc_open()`.
`version`	The library version number is returned here.
`built`	The library build date and time are returned here. The string is formatted as if produced by the below C statement. The argument may be NULL. `printf(__DATE__ ", " __TIME__);`

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

3.1.3. sio4_hdlc_open()

This function is the entry point to open a connection to an SIO4 for HDLC operation. The handle returned by this call is used for all subsequent access to the specified serial channel. The handle can be used for access via the library's high level functions, the library's low level functions, and for any access to the board that is made without the HDLC Protocol Library.

Prototype

```
int sio4_hdlc_open(int index);
```

Argument	Description
index	This is the zero based index of the SIO4 serial channel to access.

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

3.1.4. 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
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().

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

3.1.5. 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 cmd, void* arg);
```

Argument	Description
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().
cmd	This is an IOCTL macro contained in sio4.h or sio4_usc.h.
arg	This is the argument type required for the above referenced IOCTL service.

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

3.1.6. sio4_hdlc_init()

This function initializes an sio4_hdlc_t structure according to the capabilities of the accessed device and a few basic caller preferences. This function operates mostly by calling a low level function for each of the structure fields.

Prototype

```
int sio4_hdlc_init(
```

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

Argument	Description
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().
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 3.3.1, page 20.)
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	An error occurred. This is a negative errno.h value.
0	The operation completed successfully.

Data Type

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

```
typedef struct
{
    // All fields must be filled in before calling sio4_hdlc_init().
    s32         tx_bit_rate;
    s32         rx_bit_rate;
    s32         rx_uses_cbl_rxc;
} 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 10,000,000. * †
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 10,000,000. * †
rx_uses_cbl_rxc	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. †

Values	Description
SIO4_HDLC_RX_USES_CBL_RXC_NO	The receiver will get its clock from the DPLL.
SIO4_HDLC_RX_USES_CBL_RXC_YES	The receiver will get its clock from the cable's Rx Clock signal.

* If the bit rate is the same for the transmitter and the receiver, then the transmitter will use the same clock source used by the receiver.

† The cable's Tx Clock signal will be driven with the transmitter clock if the transmitter and receiver use the same bit rate, or if the receiver does not get its clock from the cable's Rx Clock signal.

3.1.7. sio4_hdlc_get()

This function retrieves the settings from the SIO4 for each of the referenced structure's fields. This function operates by calling a low level function for each of the structure fields.

Prototype

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

Argument	Description
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().
hdlc	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 3.3.1, page 20.)
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	An error occurred. This is a negative errno.h value.
0	The operation completed successfully.

3.1.8. 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. This function operates by calling a low level function for each of the structure fields.

NOTE: Before calling this function the structure should be initialized by calling the sio4_hdlc_init() function (section 3.1.6, page 11).

Prototype

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

Argument	Description
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().
hdlc	This is the structure containing the settings to be applied to the accessed device. (Refer to section 3.3.1, page 20.)
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	An error occurred. This is a negative errno.h value.
0	The operation completed successfully.

3.1.9. 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, const char** err);
```

Argument	Description
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().
hdlc	This is the structure whose content will be displayed. (Refer to section 3.3.1, page 20.)
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	An error occurred. This is a negative errno.h value.
0	The operation completed successfully.

3.1.10. 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:** An Rx Flush request is rejected and returns -EBUSY if an Rx Flush request is already active (see sio4_hdlc_rx_flush(), section 3.1.10, page 14) or if an Rx Frame request is active (see sio4_hdlc_rx_frame(), section 3.1.11, page 14).

Prototype

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

Argument	Description
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().

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

3.1.11. sio4_hdlc_rx_frame()

This function requests that an HDLC frame be read from the SIO4. If the buffer fills before the frame ends, then only a partial frame is returned. If the read timeout expires before a frame completes, then at most only a partial frame may be returned. Always consult the referenced structure's fields for completion status. This structure will always indicate the number of bytes retrieved, even if the return value of -1 indicates there was a problem accessing the device. The status flags are set by the HDLC Protocol Library and may represent either pre or post data transfer status.

> **NOTE:** An Rx Frame request is rejected and returns -EBUSY if an Rx Flush request is active (see sio4_hdlc_rx_flush(), section 3.1.10, page 14) or if an Rx Frame request is already active (see sio4_hdlc_rx_frame(), section 3.1.11, page 14).

Prototype

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

Argument	Description
fd	This is a file descriptor obtained by a call to sio4_hdlc_open().
rx	This structure provides information to the library and is where information is returned by the library. See below.

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

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
{
    // These are filled in by the caller before the transfer.
    void*       buf;
    s32         size;       // max number of bytes to get

    // These are filled in by the library after the transfer.
    s32         rcvd;       // Number of bytes transferred.
    u32         flags;      // SIO4_HDLC_FLAG_RX_*
    s32         last;       // SIO4_HDLC_RX_LAST_CHAR_LEN_*
} sio4_hdlc_rx_frame_t;
```

Field	Description
buf	This points to the buffer used for the data transfer. It is the destination buffer for Rx requests.
size	This is the size of the above buffer and represents the maximum number of bytes to read.
rcvd	This is the number of bytes received as part of the request. This is reported by the library with each frame read request. The application must initialize this to zero.
flags	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 bit values are given in the table below.

Value	Description
SIO4_HDLC_RX_FLAG_ABORT	An Abort was received.
SIO4_HDLC_RX_FLAG_CRC_ERR	A CRC error was encountered. *
SIO4_HDLC_RX_FLAG_DATA	Data was returned as part in the buffer as indicated by the rcvd field.
SIO4_HDLC_RX_FLAG_DATA_LOSS	Data for this or a following frame was lost. The specific cause is not specified. *
SIO4_HDLC_RX_FLAG_DATA_OVERRUN	There was a data overrun. Receive data was lost. *
SIO4_HDLC_RX_FLAG_DATA_UNDERRUN	Excess data was read from the Rx FIFO. Indeterminate data was returned. *
SIO4_HDLC_RX_FLAG_DPLL_CV	The DPLL lost track for one or two clock cycles. Data may have been lost. *
SIO4_HDLC_RX_FLAG_DPLL_CV1	The DPLL lost track for one clock cycle. Data may have been lost. *
SIO4_HDLC_RX_FLAG_DPLL_CV2	The DPLL lost track for two clock cycles. Data may have been lost. *
SIO4_HDLC_RX_FLAG_EOF	A completed frame was received. The last character in the buffer is the last character of the frame.
SIO4_HDLC_RX_FLAG_FRAME_OVERRUN	There was a frame record overrun. One or more frame records were lost. The frame data may have been captured, but the frame size and status are otherwise unknown. *
SIO4_HDLC_RX_FLAG_FRAME_SHORT	A frame ended before the frame content was completely received. *
SIO4_HDLC_RX_FLAG_PARITY	A parity error was detected. *

* Each of these status flags represents a condition that calls for resynchronization with the data

	stream. The current frame and some number of subsequent frames are suspect. The recovery process should include a call to `sio4_hdlc_rx_flush()`.
`last`	This is the size of the last character of the frame. This is reported by the library. If multiple calls are required to complete a frame, this is set only for the first call. This field is not modified otherwise. The application must initialize this to zero. The length specified does not include the Parity Bit, if Parity is enabled. Valid values are given in the table below.

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.

3.1.12. sio4_hdlc_tx_abort()

This function immediately initiates an Abort condition on the cable's output data signal. When waiting for completion, the library waits no longer than the specified timeout period.

NOTE: A Tx Abort request is rejected and returns `-EBUSY` if a Tx Abort request is already active (either a call or the signal at the cable interface), if a Tx Flush request is active (see `sio4_hdlc_tx_flush()`, section 3.1.13, page 16) , if a Tx Frame request is active (see `sio4_hdlc_tx_frame()`, section 3.1.14, page 17) or if a Tx Wait request is active (see `sio4_hdlc_tx_wait()`, section 3.1.15, page 19).

NOTE: Even if the wait argument indicates that the service is not to wait for completion, the abort request remains active until the abort condition is complete at the cable interface.

Prototype

```
int sio4_hdlc_tx_abort(int fd, int timeout);
```

Argument	Description
`fd`	This is a file descriptor obtained by a call to `sio4_hdlc_open()`.
`timeout`	This is the maximum amount of time, in seconds, that the call will wait for the opertion to complete at the cable interface. The valid values are from one to `3600`.

Return Value	Description
`-ETIMEDOUT`	The timeout period expired before the operation completed.
`< 0`	An error occurred. This is a negative `errno.h` value.
`0`	The operation completed successfully.

3.1.13. 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 data lost or corruption or other significant error. All buffered transmit data is discarded when this call is made. This includes data in the SIO4's Tx FIFO, and data in the USC transmitter. After the flush completes, and in the absence of any other controlling activity, the transmitter will end any frame previously in progress. After that, the transmitter will fall back to outputting its configured idle line condition. When waiting for completion, the library waits no longer than the specified timeout period.

NOTE: A Tx Flush request is rejected and returns `-EBUSY` if a Tx Abort request is active (either a call or the signal at the cable interface) (see `sio4_hdlc_tx_abort()`, section 3.1.12, page 16), if a Tx Flush request is already active, or if a Tx Frame request is active (see `sio4_hdlc_tx_frame()`, section 3.1.14, page 17).

Prototype

```
int sio4_hdlc_tx_flush(int fd, int timeout);
```

Argument	Description
`fd`	This is a file descriptor obtained by a call to `sio4_hdlc_open()`.
`timeout`	This is the maximum amount of time, in seconds, that the call will wait for the operation to complete. The valid values are from one to `3600`.

Return Value	Description
`-ETIMEDOUT`	The maximum timeout period expired before the operation completed.
`< 0`	An error occurred. This is a negative `errno.h` value.
`0`	The operation completed successfully.

3.1.14. sio4_hdlc_tx_frame()

This function requests that a block of data be transmitted as an HDLC frame. If the host cannot supply the data to the board fast enough to keep up with the Tx bit rate at the cable interface, then the data will be transmitted as two or more smaller frames. If the write timeout expires before the data provided is completely written to the board, then the application will have to perform some type of recovery operation. Always consult the referenced structure's fields for completion status. This structure will always indicate the number of bytes sent, even if the return value is less than zero. The status flags are set by the HDLC Protocol Library and may represent either pre or post data transfer status. If an error condition occurs, then the application is responsible for recovery. This recovery would normally include a call to `sio4_hdlc_tx_flush()` (see section 3.1.13, page 16). After recovery the application can begin sending new frames.

NOTE: When a frame write request completes successfully, it does not mean that the entire frame has been successfully transmitted over the cable. It only means that all frame data has been written to the board. At completion of the call the frame data may still reside entirely on the board in the channel's Tx FIFO.

NOTE: A frame write request is rejected and returns `-EBUSY` if a Tx Abort request is active (either a call or the signal at the cable interface) (see `sio4_hdlc_tx_abort()`, section 3.1.12, page 16), if a Tx Flush request is active (see `sio4_hdlc_tx_flush()`, section 3.1.13, page 16), if a Tx Frame request is already active or if a Tx Wait request is active (see `sio4_hdlc_tx_wait()`, section 3.1.15, page 19).

Prototype

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

Argument	Description
`fd`	This is a file descriptor obtained by a call to `sio4_hdlc_open()`.
`tx`	This structure provides information to the library and is where information is returned by the library. See below.

Return Value	Description
`-ETIMEDOUT`	The maximum timeout period expired before the operation completed.
`< 0`	An error occurred. This is a negative `errno.h` value.

0	The operation completed successfully.

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.
    void*       buf;
    s32         count;      // number of bytes to send
    s32         last;       // SIO4_HDLC_TX_LAST_CHAR_LEN_*

    // These are filled in by the library after the transfer.
    s32         sent;       // Number of bytes transferred.
    u32         flags;      // SIO4_HDLC_FLAG_TX_*
} sio4_hdlc_tx_frame_t;
```

Field	Description
buf	This points to the buffer used for the data transfer. It is the source for transmit data. This must be non-NULL even if the count is zero.
count	This is the number of bytes to transfer. The must be a value from zero to 0xFFFF.
last	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 not include the Parity Bit, if Parity is enabled. The active data bits for the last byte must be right justified.

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.

sent	This is the number of bytes transferred as part of the request. This does not include any bytes added as part of the frame end sequence, such as the CRC. This must be initialized to zero.
flags	This field reports status information about the transfer. Valid bit values are given in the table below. This must be initialized to zero.

Value	Description
SIO4_HDLC_TX_FLAG_EOF	This indicates that the entire frame was written to the board.
SIO4_HDLC_TX_FLAG_ERROR	This indicates that there was an unspecified error while trying to perform the operation.
SIO4_HDLC_TX_FLAG_OVERRUN	This indicates that there was a FIFO overrun.
SIO4_HDLC_TX_FLAG_UNDERRUN	This indicates that the frame completed prematurely.

3.1.15. sio4_hdlc_tx_wait()

This function initiates a wait for completion of all the following. The wait will continue so long as any of the following are still active, but no longer than the specified timeout limit. Multiple wait requests can be active simultaneously, and each operates independently.

1. It will wait while a Tx Abort request is still active (either a call or the signal at the cable interface) (see `sio4_hdlc_tx_abort()`, section 3.1.12, page 16).

2. It will wait while a Tx Frame request is still active (see `sio4_hdlc_tx_frame()`, section 3.1.14, page 17). This refers only to the period where the call is active.

3. It will wait while there is data in the external Tx FIFO or the USC internal Tx FIFO.

4. It will wait while data is being sent out the cable interface.

Prototype

```
int sio4_hdlc_tx_wait(int fd, int timeout, int* done);
```

Argument	Description
`fd`	This is a file descriptor obtained by a call to `sio4_hdlc_open()`.
`timeout`	This is the maximum amount of time, in seconds, that the call will wait for the operation to complete. The valid values are from zero to `3600`.
`done`	The library reports here if the transmitter finished sending out all data. This will be zero if the transmitter did not finish and non-zero if it did. This pointer must not be NULL.

Return Value	Description
`-ETIMEDOUT`	The maximum timeout period expired before the operation completed.
`< 0`	An error occurred. This is a negative `errno.h` value.
`0`	The operation completed successfully.

3.2. Low Level Functions

The low level functions provide access to the board features reflected by the individual fields of the `sio4_hdlc_t` structure (see section 3.3.1, page 20). This structure is used to access all of the board features that are part of the HDLC Protocol Library. For each structure field there is a corresponding board feature and an associated low level function. When calling the high level functions that use the `sio4_hdlc_t` structure, the high level functions perform their work by calling the low level functions for each of the structure's fields. This is especially useful for activities such as structure initialization and board configuration. Following high level configuration of the board though, there are times when an application may need to access features represented by only one or two the `sio4_hdlc_t` structure fields. This is where an application can make use of the low level function. All of the low level functions follow the prototype pattern shown below. The function name includes the prefix "`sio4_hdlc_t_`" followed by the C style field name, but with the periods ("`.`") replaced by underscores ("`_`").

Prototype

```
void sio4_hdlc_t_field_name(int fd, s32* arg,
    sio4_hdlc_action_t action, const char** err);
```

Argument	Description
`fd`	This is a file descriptor obtained by a call to `sio4_hdlc_open()`.
`arg`	This refers to the feature specific value being passed in to the function. This pointer must

	not be NULL.
action	This identifies the specific action to be carried out in regards to the above feature specific value. See below.
err	If there is an error, then this field will be set to point to a string naming the sio4_hdlc_t field for which the error pertains. An example is "hdlc-> cable.loopback.mode". This pointer must not be NULL.

Data Type

This enumeration identifies the specific actions called for when a low level function is called.

```
typedef enum
{
    SIO4_HDLC_ACTION_GET,
    SIO4_HDLC_ACTION_INIT,
    SIO4_HDLC_ACTION_SET,
    SIO4_HDLC_ACTION_SHOW,
    SIO4_HDLC_ACTION_VERIFY
} sio4_hdlc_action_t;
```

Field	Description
SIO4_HDLC_ACTION_GET	This requests the current setting from the driver.
SIO4_HDLC_ACTION_INIT	This requests the initialization value from the library.
SIO4_HDLC_ACTION_SET	This requests that the supplied value be applied by the driver.
SIO4_HDLC_ACTION_SHOW	This requests that the supplied value be displayed to the screen.
SIO4_HDLC_ACTION_VERIFY	This requests that the supplied value be verified.

Examples

For simplicity sake a low level function name can easily be derived given any field name, as shown in the below examples. The individual low level function names are identified with the corresponding structure fields beginning in section 3.3.1, page 20.

Field	Function
sio4_hdlc_t.cable.loopback.mode	sio4_hdlc_t_cable_loopback_mode()
sio4_hdlc_t.rx.encoding	sio4_hdlc_t_rx_encoding()
sio4_hdlc_t.tx.preamble.pattern	sio4_hdlc_t_tx_preamble_pattern()

3.3. 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.

3.3.1. 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() function (section 3.1.6, page 11). 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 3.1.8, page 13) where the setting are applied to the board.

```
typedef struct
{
    // All fields are filled in by the library when calling
    // sio4_hdlc_init(). Application mods must be made before calling
    // sio4_hdlc_set(). sio4_hdlc_init() will set a field to -1
    // if the feature is unsupported.

    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 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     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 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 folling USC fields are initialized
    // by sio4_hdlc_init() 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 TC0R 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 + ...
        } ctr1;

        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;
```

3.3.1.1. 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. The feature's low level function is `sio4_hdlc_t_osc_ref()`.
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. The feature's low level function is `sio4_hdlc_t_osc_prog()`.

3.3.1.2. sio4_hdlc_t.cable

This section describes the structure's cable configuration fields.

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>cable</td><td>This structure configures the cable interface.</td></tr>
<tr><td>cable.
enable</td><td>This field either enables or disables the cable transceivers. Valid values are given in the table below. The feature's low level function is <code>sio4_hdlc_t_cable_enable()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_CABLE_ENABLE_NO</td><td>Leave the cable transceivers disabled.</td></tr>
<tr><td>SIO4_HDLC_CABLE_ENABLE_YES</td><td>Enable the cable transceiver. This is the default.</td></tr>
</table></td></tr>
<tr><td>cable.
mode</td><td>This field specifies the arrangement of the signals on the cable interface. Valid values are given in the table below. The feature's low level function is <code>sio4_hdlc_t_cable_mode()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_CABLE_MODE_DCE</td><td>Select the DCE cable signal configuration.</td></tr>
<tr><td>SIO4_HDLC_CABLE_MODE_DTE</td><td>Select the DTE cable signal configuration. This is the default.</td></tr>
</table></td></tr>
<tr><td>cable.
protocol</td><td>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. The feature's low level function is <code>sio4_hdlc_t_cable_protocol()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_DISABLE</td><td>This disables the cable transceivers.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_RS232</td><td>This selects the RS232 protocol. This is the default.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_RS422_423_1</td><td>This selects the RS422/RS423 mixed protocol version 1.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_RS422_423_2</td><td>This selects the RS422/RS423 mixed protocol version 2.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_RS422_RS485</td><td>This selects the RS422/RS485 mixed protocol.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_RS423</td><td>This selects the RS423 protocol.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_RS530</td><td>This selects the RS530 protocol, version 1.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_RS530A</td><td>This selects the RS530 protocol, version 2.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_V35</td><td>This selects the V.35 protocol, version 1.</td></tr>
<tr><td>SIO4_HDLC_CABLE_PROTOCOL_V35A</td><td>This selects the V.35 protocol, version 2.</td></tr>
</table></td></tr>
<tr><td>cable.
txc</td><td>This field specifies the configuration of the cable's Tx Clock signal. Valid values are given in the table below. The feature's low level function is <code>sio4_hdlc_t_cable_txc()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_CABLE_TXC_OUT_0</td><td>This drives the signal low.</td></tr>
</table></td></tr>
</table>

	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. The feature's low level function is `sio4_hdlc_t_cable_txd()`.

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.txaux

This field specifies the configuration of the cable's Tx Aux signal. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_cable_txaux()`.

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

This field specifies the cable DCD signal source when the cable signal is driven. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_cable_dcd()`.

NOTE: Refer to the `usc.dcd` field (section 3.3.1.5, page 37) for affecting the cable signal's driven state.

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

This field specifies the configuration of the cable's DTR/DSR signal. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_cable_dtr_dsr()`.

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

This field specifies the configuration of the cable's RTS signal. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_cable_rts()`.

	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	This field specifies the loopback mode. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_cable_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. * If external loopback mode is requested but not available, then the internal loopback mode is selected.
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. The feature's low level function is sio4_hdlc_t_cable_term_enable(). **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.
cable.legacy.txc	This field specifies the legacy configuration of the cable's Tx Clock signal. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_cable_legacy_txc(). **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. The feature's low level function is sio4_hdlc_t_cable_legacy_txd_cts(). **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. The feature's low level function is `sio4_hdlc_t_cable_legacy_rxc()`.

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. The feature's low level function is `sio4_hdlc_t_cable_legacy_rxd_dcd()`.

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.

3.3.1.3. 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	This field specifies the transmitter's operating mode. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_tx_mode()`.

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	This field specifies if the transmitter is to be enabled. When configuration is begun (see `sio4_hdlc_set()`, section 3.1.8, page 13) 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. The feature's low level function is `sio4_hdlc_t_tx_enable()`.

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	This field specifies if the size of transmitted characters. The length specified includes 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. The feature's low level function is `sio4_hdlc_t_tx_char_len()`.

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`

This field specifies if the encoding of the transmitted data. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_tx_encoding()`.

Value	Description
SIO4_HDLC_TX_ENCODING_BI_MARK	This refers to Biphase Mark encoding.
SIO4_HDLC_TX_ENCODING_BI_LEVEL	This refers to Biphase Level encoding.
SIO4_HDLC_TX_ENCODING_BI_SPACE	This refers to Biphase Space encoding.
SIO4_HDLC_TX_ENCODING_D_BI_LEVEL	This refers to Differential Biphase 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 `sio4_hdlc_init()` call (section 3.1.6, page 11) this is computed from the `sio4_hdlc_init_t.tx_bit_rate` field provided to the call. The feature's low level function is `sio4_hdlc_t_tx_bit_rate()`.

`tx.idle_cond`

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. The feature's low level function is `sio4_hdlc_t_tx_idle_cond()`.

Value	Description
SIO4_HDLC_TX_IDLE_COND_0	The Tx Data signal is driven low.
SIO4_HDLC_TX_IDLE_COND_0_1	The Tx Data signal is alternately driven low then high.
SIO4_HDLC_TX_IDLE_COND_1	The Tx Data signal is driven low.
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 driven with the Mark state.
SIO4_HDLC_TX_IDLE_COND_MARK_SPACE	The Tx Data signal is alternately driven with the Mark and Space states.
SIO4_HDLC_TX_IDLE_COND_SPACE	The Tx Data signal is driven with the Space state.

`tx.share_0`

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. The feature's low level function is `sio4_hdlc_t_tx_share_0()`.

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.

<table>
<tr><td>tx.
underrun</td><td>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. The feature's low level function is sio4_hdlc_t_tx_underrun().
<table>
<tr><th>Value</th><th>Description</th></tr>
<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>
</table></td></tr>
<tr><td>tx.
wait_underrun</td><td>This field specifies the transmitter's reaction to running out of data when additional data is needed to complete a frame. The feature's low level function is sio4_hdlc_t_tx_wait_underrun().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_TX_WAIT_UNDERRUN_NO</td><td>The transmitter is to end the frame prematurely rather than wait for additional data. This is the default.</td></tr>
<tr><td>SIO4_HDLC_TX_WAIT_UNDERRUN_YES</td><td>The transmitter is to wait for additional data and not end the frame prematurely.</td></tr>
</table></td></tr>
<tr><td>tx.
crc</td><td>This structure configures the transmitter's use of a CRC at the end of a Frame.</td></tr>
<tr><td>tx.
crc.
enable</td><td>This field enables or disables use of a CRC at the end of frames. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_crc_enable().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_TX_CRC_ENABLE_NO</td><td>CRCs are not used.</td></tr>
<tr><td>SIO4_HDLC_TX_CRC_ENABLE_YES</td><td>CRCs are used. This is the default.</td></tr>
</table></td></tr>
<tr><td>tx.
crc.
type</td><td>This field selects the type of CRC used, when CRC use is enabled. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_crc_type().
<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></td></tr>
<tr><td>tx.
crc.
preset</td><td>This field selects the CRC starting value. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_crc_preset().
<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></td></tr>
<tr><td>tx.
crc.
on_end</td><td>This field specifies if a CRC is to be send at the end of a Frame. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_crc_on_end().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_TX_CRC_ON_END_NO</td><td>Do not 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></td></tr>
<tr><td>tx.
parity</td><td>This structure configures the transmitter's use of Parity checking.</td></tr>
<tr><td>tx.
parity.</td><td>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,</td></tr>
</table>

<table>
<tr><td>enable</td><td>the Parity Bit appears to the immediate left of the most significant data bit. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_parity_enable().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_TX_PARITY_ENABLE_NO</td><td>Do not 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></td></tr>
<tr><td>tx.
parity.
type</td><td>This field specifies the type of Parity to use, when its use is enabled. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_parity_type().
<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></td></tr>
<tr><td>tx.
preamble</td><td>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.</td></tr>
<tr><td>tx.
preamble.
enable</td><td>This field enables or disables the use of a Preamble sequence. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_preamble_enable().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_TX_PREAMBLE_ENABLE_NO</td><td>Do not 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></td></tr>
<tr><td>tx.
preamble.
flag</td><td>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. The feature's low level function is sio4_hdlc_t_tx_preamble_flag().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_TX_PREAMBLE_FLAG_NO</td><td>Do not use the Flag sequence.</td></tr>
<tr><td>SIO4_HDLC_TX_PREAMBLE_FLAG_YES</td><td>Do use the Flag sequence. This is the default.</td></tr>
</table></td></tr>
<tr><td>tx.
preamble.
pattern</td><td>This field selects the Preamble Pattern to use, when use of a Preamble is enabled. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_preamble_pattern().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_TX_PREAMBLE_PATTERN_0</td><td>This specifies continuous zero bits.</td></tr>
<tr><td>SIO4_HDLC_TX_PREAMBLE_PATTERN_1</td><td>This specifies continuous one bits. If the above Flag option is Yes, then this option refers to the Flag sequence. This is the default.</td></tr>
<tr><td>SIO4_HDLC_TX_PREAMBLE_PATTERN_01</td><td>This specifies a pattern of a zero bit followed by a one bit.</td></tr>
<tr><td>SIO4_HDLC_TX_PREAMBLE_PATTERN_10</td><td>This specifies a pattern of a one bit followed by a zero bit.</td></tr>
</table></td></tr>
<tr><td>tx.</td><td>This field specifies the length of the Preamble Pattern to use, when use of a Preamble is</td></tr>
</table>

<table>
<tr><td>preamble.
length</td><td>enabled. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_preamble_length().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_TX_PREAMBLE_LENGTH_8_BITS</td><td>The length is 8-bits. This is the default.</td></tr><tr><td>SIO4_HDLC_TX_PREAMBLE_LENGTH_16_BITS</td><td>The length is 16-bits.</td></tr><tr><td>SIO4_HDLC_TX_PREAMBLE_LENGTH_32_BITS</td><td>The length is 32-bits.</td></tr><tr><td>SIO4_HDLC_TX_PREAMBLE_LENGTH_64_BITS</td><td>The length is 64-bits.</td></tr></table></td></tr>
<tr><td>tx.
fifo</td><td>This structure configures the transmitter's FIFO parameters.</td></tr>
<tr><td>tx.
fifo.
size</td><td>This field is filled in by the sio4_hdlc_init() call (section 3.1.6, page 11) with the size of the channel's Tx FIFO. This is offered for informational purposes only. The feature's low level function is sio4_hdlc_t_tx_fifo_size().</td></tr>
<tr><td>tx.
fifo.
ae</td><td>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. The feature's low level function is sio4_hdlc_t_tx_fifo_ae().</td></tr>
<tr><td>tx.
fifo.
af</td><td>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. The feature's low level function is sio4_hdlc_t_tx_fifo_af().</td></tr>
<tr><td>tx.
fifo.
empty_cfg</td><td>This field configures the transmitter's reaction to the Tx FIFO becoming empty. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_tx_fifo_empty_cfg().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_TX_FIFO_EMPTY_CFG_IGNORE</td><td>This specifies that the condition is to be ignored. This is the default.</td></tr><tr><td>SIO4_HDLC_TX_FIFO_EMPTY_CFG_TX_OFF</td><td>This specifies that the transmitter be disabled when the condition occurs.</td></tr></table></td></tr>
<tr><td>tx.
fifo.
space_cfg</td><td>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. The feature's low level function is sio4_hdlc_t_tx_fifo_space_cfg().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_TX_FIFO_SPACE_CFG_RX_2X</td><td>This specifies that the Rx FIFO be twice as large as the Tx FIFO. This is the default.</td></tr><tr><td>SIO4_HDLC_TX_FIFO_SPACE_CFG_TX_2X</td><td>This specifies that the Tx FIFO be twice as large as the Rx FIFO.</td></tr></table></td></tr>
<tr><td>tx.
io</td><td>This structure configures the transmitter's software settings. These settings are used during sio4_hdlc_tx_frame() calls (see section 3.1.14, page 17).</td></tr>
<tr><td>tx.
io.
mode</td><td>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. The feature's low level function is sio4_hdlc_t_tx_io_mode().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_TX_IO_MODE_DMA</td><td>This selects DMA mode transfers. *</td></tr><tr><td>SIO4_HDLC_TX_IO_MODE_DMDMA</td><td>This selects Demand Mode DMA mode transfers. *</td></tr><tr><td>SIO4_HDLC_TX_IO_MODE_PIO</td><td>This selects PIO mode transfers. This is the default.</td></tr></table>* The SIO4 has only two DMA engines. A DMA or DMDMA transfer request will fail if both DMA engines are already in use by other SIO4 channels.</td></tr>
</table>

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 64. The feature's low level function is sio4_hdlc_t_tx_io_pio_thresh().
tx. io. timeout	This field specifies the maximum duration of write requests to the driver. This refers to calls made to the driver and not to calls made to the HDLC Protocol Library. The valid range is from zero to 3600. The units are seconds. The value zero should be used with PIO mode only as it tells the driver to write as much data as possible to the Tx FIFO, but not to wait for addition free space. DMA and DMDMA requests always require a sleep to wait for the hardware to complete the transfer. The default is 10 seconds. The feature's low level function is sio4_hdlc_t_tx_io_timeout().
tx. io. overrun	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. The feature's low level function is sio4_hdlc_t_tx_io_overrun().

Value	Description
SIO4_HDLC_TX_IO_OVERRUN_CHECK	This specifies that the driver should check for overrun conditions.
SIO4_HDLC_TX_IO_OVERRUN_IGNORE	This specifies that the driver should not check for overrun conditions. This is the default. Overrun testing is performed by the library and need not be performed by the driver.

3.3.1.4. 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	This field specifies the receiver's operating mode. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_rx_mode().

Value	Description
SIO4_HDLC_RX_MODE_HDLC	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. The feature's low level function is sio4_hdlc_t_rx_adrs().
rx. adrs_ctrl	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. The feature's low level function is sio4_hdlc_t_rx_adrs_ctrl().

Value	Description
SIO4_HDLC_RX_ADRS_CTRL_16	The field is fixed at 16-bits. This is the default.
SIO4_HDLC_RX_ADRS_CTRL_24	The field is fixed at 24-bits.
SIO4_HDLC_RX_ADRS_CTRL_32	The field is fixed at 32-bits.
SIO4_HDLC_RX_ADRS_CTRL_EA_16	The address field is variable length, but the control field is fixed at 16-bits.
SIO4_HDLC_RX_ADRS_CTRL_EA_24	The address field is variable length, but the control field is fixed at 24-bits.
SIO4_HDLC_RX_ADRS_CTRL_EAC8	The address and control field are variable length, and are followed by a fixed 8-bit field.
SIO4_HDLC_RX_ADRS_CTRL_EAC16	The address and control field are variable

		length, and are followed by a fixed 16-bit field.
	SIO4_HDLC_RX_ADRS_CTRL_OFF	No address or control fields are included.

rx. enable

This field specifies if the receiver is to be enabled. When configuration is begun (see sio4_hdlc_set(), section 3.1.8, page 13) 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. The feature's low level function is sio4_hdlc_t_rx_enable().

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 includes 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. The feature's low level function is sio4_hdlc_t_rx_char_len().

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. The feature's low level function is sio4_hdlc_t_rx_encoding().

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. bit_rate

This specifies the desired receive data bit rate. During the sio4_hdlc_init() call (section 3.1.6, page 11) this is computed from the sio4_hdlc_init_t.rx_bit_rate field provided to the call. The feature's low level function is sio4_hdlc_t_rx_bit_rate().

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. The feature's low level function is sio4_hdlc_t_rx_queue_abort().

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

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>USC Rx FIFO. This is the default. †</td></tr>
</table>
* 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.
</td></tr>
<tr><td>rx.
sync_byte</td><td>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. The feature's low level function is sio4_hdlc_t_rx_sync_byte().</td></tr>
<tr><td>rx.
status_word</td><td>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. The feature's low level function is sio4_hdlc_t_rx_status_word().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_RX_STATUS_WORD_DISABLE</td><td>The RCSR is not placed in the Rx FIFO. This is the default.</td></tr>
<tr><td>SIO4_HDLC_RX_STATUS_WORD_ENABLE</td><td>The RCSR is placed in the Rx FIFO. *</td></tr>
</table>
* 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.</td></tr>
<tr><td>rx.
crc</td><td>This structure configures the receiver's use of a CRC at the end of a Frame.</td></tr>
<tr><td>rx.
crc.
enable</td><td>This field enables or disables use of a CRC at the end of frames. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_rx_crc_enable().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_RX_CRC_ENABLE_NO</td><td>CRCs are not used.</td></tr>
<tr><td>SIO4_HDLC_RX_CRC_ENABLE_YES</td><td>CRCs are used. This is the default.</td></tr>
</table></td></tr>
<tr><td>rx.
crc.
type</td><td>This field selects the type of CRC used, when CRC use is enabled. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_rx_crc_type().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_RX_CRC_TYPE_16</td><td>This selects the 16-bit polynomial CRC.</td></tr>
<tr><td>SIO4_HDLC_RX_CRC_TYPE_32</td><td>This selects the 32-bit polynomial CRC.</td></tr>
<tr><td>SIO4_HDLC_RX_CRC_TYPE_CCITT</td><td>This selects the 16-bit CCITT CRC. This is the default.</td></tr>
</table></td></tr>
<tr><td>rx.
crc.
preset</td><td>This field selects the CRC starting value. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_rx_crc_preset().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_RX_CRC_PRESET_ALL_0</td><td>Use a starting value of all zeroes.</td></tr>
<tr><td>SIO4_HDLC_RX_CRC_PRESET_ALL_1</td><td>Use a starting value of all ones. This is the default.</td></tr>
</table></td></tr>
<tr><td>rx.
parity</td><td>This structure configures the receiver's use of Parity checking.</td></tr>
<tr><td>rx.
parity.
enable</td><td>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. The feature's low level function is sio4_hdlc_t_rx_parity_enable().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_RX_PARITY_ENABLE_NO</td><td>Do not generate a Parity bit. This is the default.</td></tr>
<tr><td>SIO4_HDLC_RX_PARITY_ENABLE_YES</td><td>Do generate a Parity bit.</td></tr>
</table></td></tr>
<tr><td>rx.</td><td>This field specifies the type of Parity to use, when its use is enabled. Valid values are given in</td></tr>
</table>

<table>
<tr><td>parity.
type</td><td>the table below. The feature's low level function is sio4_hdlc_t_rx_parity_type().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_RX_PARITY_TYPE_EVEN</td><td>This specifies Even Parity. This is the default.</td></tr><tr><td>SIO4_HDLC_RX_PARITY_TYPE_ODD</td><td>This specifies Odd Parity.</td></tr><tr><td>SIO4_HDLC_RX_PARITY_TYPE_ONE</td><td>This specifies One Parity (the parity bit is always set).</td></tr><tr><td>SIO4_HDLC_RX_PARITY_TYPE_ZERO</td><td>This specifies Zero Parity (the parity bit is always clear).</td></tr></table></td></tr>
<tr><td>rx.
fifo</td><td>This structure configures the receiver's FIFO parameters.</td></tr>
<tr><td>rx.
fifo.
size</td><td>This field is filled in by the sio4_hdlc_init() call (section 3.1.6, page 11) with the size of the channel's Rx FIFO. This is offered for informational purposes only. The feature's low level function is sio4_hdlc_t_rx_fifo_size().</td></tr>
<tr><td>rx.
fifo.
ae</td><td>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. The feature's low level function is sio4_hdlc_t_rx_fifo_ae().</td></tr>
<tr><td>rx.
fifo.
af</td><td>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. The feature's low level function is sio4_hdlc_t_rx_fifo_af().</td></tr>
<tr><td>rx.
fifo.
full_cfg</td><td>This field configures the receiver's reaction to the Rx FIFO becoming full. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_rx_fifo_full_cfg(). 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.
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_RX_FIFO_FULL_CFG_DISABLE</td><td>This specifies that the receiver be disabled when the condition occurs.</td></tr><tr><td>SIO4_HDLC_RX_FIFO_FULL_CFG_OVER</td><td>This specifies that the condition be ignored. This is the default.</td></tr></table></td></tr>
<tr><td>rx.
io</td><td>This structure configures the receiver's software settings. These settings are used during sio4_hdlc_rx_frame() calls (section 3.1.11, page 14).</td></tr>
<tr><td>rx.
io.
mode</td><td>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. The feature's low level function is sio4_hdlc_t_rx_io_mode().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_RX_IO_MODE_DMA</td><td>This selects DMA mode transfers. *</td></tr><tr><td>SIO4_HDLC_RX_IO_MODE_DMDMA</td><td>This selects Demand Mode DMA mode transfers. *</td></tr><tr><td>SIO4_HDLC_RX_IO_MODE_PIO</td><td>This selects PIO mode transfers. This is the default.</td></tr></table>* The SIO4 has only two DMA engines. A DMA or DMDMA transfer request will fail if both DMA engines are already in use by other SIO4 channels.</td></tr>
<tr><td>rx.
io.
pio_thresh</td><td>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 64. The feature's low level function is sio4_hdlc_t_rx_io_pio_thresh().</td></tr>
<tr><td>rx.
io.
timeout</td><td>This field specifies the maximum duration of a call to the driver when reading data from the board. This refers to calls made to the driver and not to calls made to the HDLC Protocol Library. The valid range is from zero to 3600. The units are seconds. The value zero should be used with PIO mode only as it tells the driver to read as much data as possible from the Rx FIFO, but not to wait for addition data. DMA and DMDMA requests always require a sleep to wait for the hardware to complete the transfer. The default is 10 seconds. The feature's low</td></tr>
</table>

<table>
<tr><td></td><td>level function is <code>sio4_hdlc_t_rx_io_timeout()</code>.</td></tr>
<tr><td><code>rx.
io.
overrun</code></td><td>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. The feature's low level function is <code>sio4_hdlc_t_rx_io_overrun()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>SIO4_HDLC_RX_IO_OVERRUN_CHECK</code></td><td>This specifies that the driver should check for overrun conditions.</td></tr>
<tr><td><code>SIO4_HDLC_RX_IO_OVERRUN_IGNORE</code></td><td>This specifies that the driver should not 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></td></tr>
<tr><td><code>rx.
io.
underrun</code></td><td>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. The feature's low level function is <code>sio4_hdlc_t_rx_io_underrun()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>SIO4_HDLC_RX_IO_UNDERRUN_CHECK</code></td><td>This specifies that the driver should check for underrun conditions.</td></tr>
<tr><td><code>SIO4_HDLC_RX_IO_UNDERRUN_IGNORE</code></td><td>This specifies that the driver should not check for underrun conditions. This is the default. Underrun testing is performed by the library and need not be performed by the driver.</td></tr>
</table></td></tr>
<tr><td><code>rx.
time_stamp</code></td><td>This structure configures the receiver's Time Stamp settings.</td></tr>
<tr><td><code>rx.
time_stamp.
enable</code></td><td>This field enables or disables the channels use of the Time Stamp feature. Valid values are given in the table below. The feature's low level function is <code>sio4_hdlc_t_rx_time_stamp_enable()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>SIO4_HDLC_RX_TIME_STAMP_ENABLE_NO</code></td><td>Do not use the Time Stamp feature. This is the default.</td></tr>
<tr><td><code>SIO4_HDLC_RX_TIME_STAMP_ENABLE_YES</code></td><td>Do use the Time Stamp feature. *</td></tr>
</table>
* The HDLC Protocol Library does not accommodate reading Frames with this enabled.</td></tr>
<tr><td><code>rx.
time_stamp.
clk_src</code></td><td>This field selects the Time Stamp clock source. Valid values are given in the table below. The feature's low level function is <code>sio4_hdlc_t_rx_time_stamp_clk_src()</code>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>SIO4_HDLC_RX_TIME_STAMP_CLK_SRC_EXT</code></td><td>Use the board's external TTL clock source. *</td></tr>
<tr><td><code>SIO4_HDLC_RX_TIME_STAMP_CLK_SRC_INT</code></td><td>Use the board's internal 1us clock. This is the default. *</td></tr>
</table>
* All four channels on the SIO4 use the same clock source.</td></tr>
</table>

3.3.1.5. 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.`	This field specifies the USC's overall operating mode. Valid values are given in the table below.

mode

The feature's low level function is `sio4_hdlc_t_usc_mode()`.

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

This field configures the operation of the USC's Tx Data pin. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_usc_txd()`.

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

This field configures the operation of the USC's CTS pin. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_usc_cts()`.

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

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. The feature's low level function is `sio4_hdlc_t_usc_cts_legacy()`.

Value	Description
SIO4_HDLC_USC_TX_CTS_LEG_IN	The pin operates as an input. This is the default.
SIO4_HDLC_USC_TX_CTS_LEG_OUT_0	The pin operates as an output driven low.
SIO4_HDLC_USC_TX_CTS_LEG_OUT_1	The pin operates as an output driven high.

usc.dcd

This field configures the operation of the USC's DCD pin. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_usc_dcd()`.

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. *

* This option enables the cable DCD signal to be driven, though the `cable.dcd` field (section 3.3.1.2, page 25) may configure the cable to output an alternate signal.

`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. The feature's low level function is `sio4_hdlc_t_usc_dcd_legacy()`.

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`

This field configures the source for the USC transmitter clock. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_usc_tx_clk_src()`.

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.

* This is the initial default, though it may change to satisfy bit rate requirements.

`usc.tx.txc`

This field configures the operation of the USC's Tx Clock pin. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_usc_tx_txc()`.

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.

* This is the initial default, though it may change to satisfy bit rate requirements.

`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. The feature's low level function is `sio4_hdlc_t_usc_tx_txc_legacy()`.

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. The feature's low level function is `sio4_hdlc_t_usc_rx_clk_src()`.

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. The feature's low level function is `sio4_hdlc_t_usc_rx_rxc()`.

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. The feature's low level function is `sio4_hdlc_t_usc_rx_rxc_legacy()`.

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. The feature's low level function is `sio4_hdlc_t_usc_brg0_enable()`.

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. The feature's low level function is `sio4_hdlc_t_usc_brg0_clk_src()`.

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. The feature's low level function is `sio4_hdlc_t_usc_brg0_divider()`.

`usc.brg0.mode`

This field specifies the BRG0 operating mode. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_usc_brg0_mode()`.

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. The feature's low level function is `sio4_hdlc_t_usc_brg1_enable()`.

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. The feature's low level function is `sio4_hdlc_t_usc_brg1_clk_src()`.

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. The feature's low level function is `sio4_hdlc_t_usc_brg1_divider()`.

`usc.brg1.mode`

This field specifies the BRG1 operating mode. Valid values are given in the table below. The feature's low level function is `sio4_hdlc_t_usc_brg1_mode()`.

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. The feature's low level function is `sio4_hdlc_t_usc_ctr0_clk_src()`.

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.`

This field selects the divider rate for CTR0. Valid values are given in the table below. The

<table>
<tr><td>rate</td><td>feature's low level function is sio4_hdlc_t_usc_ctr0_rate().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_USC_CTR0_RATE_4X</td><td>This sets the output as the input divided by four. *</td></tr><tr><td>SIO4_HDLC_USC_CTR0_RATE_8X</td><td>This sets the output as the input divided by eight.</td></tr><tr><td>SIO4_HDLC_USC_CTR0_RATE_16X</td><td>This sets the output as the input divided by 16.</td></tr><tr><td>SIO4_HDLC_USC_CTR0_RATE_32X</td><td>This sets the output as the input divided by 32.</td></tr></table>* This is the initial default, though it may change to satisfy bit rate requirements.</td></tr>
<tr><td>usc.
ctr1</td><td>This structure configures a few of the settings for Counter 1 (CTR1).</td></tr>
<tr><td>usc.
ctr1.
clk_src</td><td>This field selects the clock source for CTR1. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_usc_ctr1_clk_src().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_USC_CTR1_CLK_SRC_DISABLE</td><td>This disables CTR1. *</td></tr><tr><td>SIO4_HDLC_USC_CTR1_CLK_SRC_RXC_PIN</td><td>This selects the signal present at the USC's Rx Clock pin.</td></tr><tr><td>SIO4_HDLC_USC_CTR1_CLK_SRC_TXC_PIN</td><td>This selects the signal present at the USC's Tx Clock pin.</td></tr></table>* This is the initial default, though it may change to satisfy bit rate requirements.</td></tr>
<tr><td>usc.
ctr1.
rate_src</td><td>This field selects the source for the rate divider used by CTR1. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_usc_ctr1_rate_src().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_USC_CTR1_RATE_SRC_CTR0</td><td>This selects the rate divider used by CTR0. *</td></tr><tr><td>SIO4_HDLC_USC_CTR1_RATE_SRC_DPLL</td><td>This selects the rate divider used by the DPLL.</td></tr></table>* This is the initial default, though it may change to satisfy bit rate requirements.</td></tr>
<tr><td>usc.
dpll</td><td>This structure configures a few of the settings for the DPLL.</td></tr>
<tr><td>usc.
dpll.
clk_src</td><td>This field selects the clock source for the DPLL. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_usc_dpll_clk_src().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_USC_DPLL_CLK_SRC_BRG0</td><td>This selects the output from BRG0. *</td></tr><tr><td>SIO4_HDLC_USC_DPLL_CLK_SRC_BRG1</td><td>This selects the output from BRG1.</td></tr><tr><td>SIO4_HDLC_USC_DPLL_CLK_SRC_RXC_PIN</td><td>This selects the signal present at the USC's Rx Clock pin.</td></tr><tr><td>SIO4_HDLC_USC_DPLL_CLK_SRC_TXC_PIN</td><td>This selects the signal present at the USC's Tx Clock pin.</td></tr></table>* This is the initial default, though it may change to satisfy bit rate requirements.</td></tr>
<tr><td>usc.
dpll.
mode</td><td>This field specifies the DPLL operating mode, which corresponds to the Rx Data Encoding format. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_usc_dpll_mode().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_HDLC_USC_DPLL_MODE_BIPH_LVL</td><td>This refers to Biphase-Level.</td></tr><tr><td>SIO4_HDLC_USC_DPLL_MODE_BIPH_MS</td><td>This refers to either Biphase-Mark or Biphase Space.</td></tr><tr><td>SIO4_HDLC_USC_DPLL_MODE_DISABLE</td><td>This disables the DPLL. *</td></tr><tr><td>SIO4_HDLC_USC_DPLL_MODE_NRZ_NRZI</td><td>This refers to either NRZ or NRZI.</td></tr></table>* This is the initial default, though it may change to satisfy bit rate requirements.</td></tr>
<tr><td>usc.
dpll.</td><td>This field selects the divider rate for the DPLL. Valid values are given in the table below. The</td></tr>
</table>

<table>
<tr><td>rate</td><td>feature's low level function is sio4_hdlc_t_usc_dpll_rate().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_USC_DPLL_RATE_CTR1_4X</td><td>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.</td></tr>
<tr><td>SIO4_HDLC_USC_DPLL_RATE_8X</td><td>This sets the output as the input divided by eight. *</td></tr>
<tr><td>SIO4_HDLC_USC_DPLL_RATE_16X</td><td>This sets the output as the input divided by 16.</td></tr>
<tr><td>SIO4_HDLC_USC_DPLL_RATE_32X</td><td>This sets the output as the input divided by 32.</td></tr>
</table>
* This is the initial default, though it may change to satisfy bit rate requirements.</td></tr>
<tr><td>usc.
dpll.
edge</td><td>This field selects the source signal edges that the DPLL uses for synchronization. Valid values are given in the table below. The feature's low level function is sio4_hdlc_t_usc_dpll_edge().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_HDLC_USC_DPLL_EDGE_BOTH_EDGE</td><td>This selects both rising and falling edges that the DPLL is to use for clocking synchronization.</td></tr>
<tr><td>SIO4_HDLC_USC_DPLL_EDGE_FALL_EDGE</td><td>This selects the falling edges that the DPLL is to use for clocking synchronization.</td></tr>
<tr><td>SIO4_HDLC_USC_DPLL_EDGE_INHIBIT</td><td>This inhibits the DPLL from synchronizing on the input clock. *</td></tr>
<tr><td>SIO4_HDLC_USC_DPLL_EDGE_RISE_EDGE</td><td>This selects the rising edges that the DPLL is to use for clocking synchronization.</td></tr>
</table>
* This is the initial default, though it may change to satisfy bit rate requirements.</td></tr>
</table>

4. Operation

This section is intended to provide limited information on the operation of the board and/or the HDLC Protocol Library.

4.1. 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.

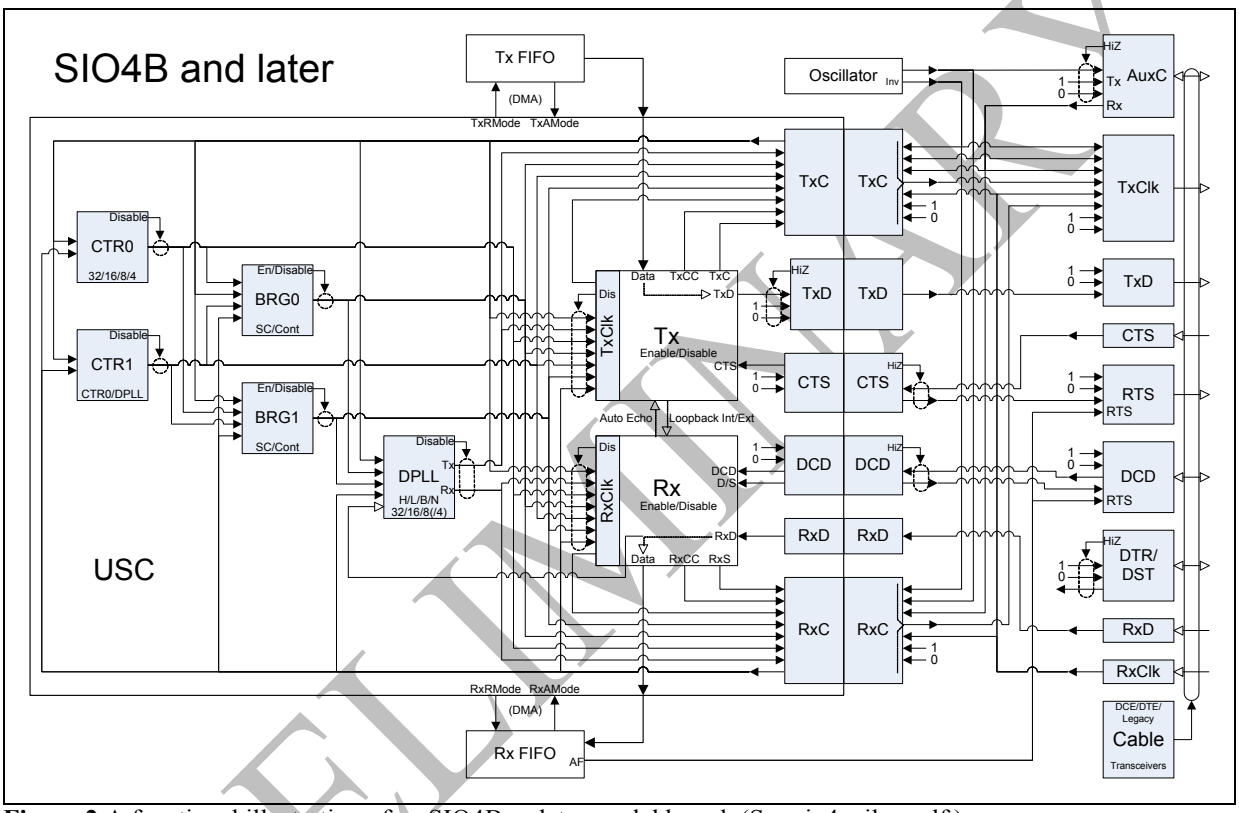

Figure 2 A functional illustration of an SIO4B or later model board. (See sio4_zilog.pdf.)

4.2. Clocking Configurations

The function `sio4_hdlc_init()` 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 results 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 yellow dashed lines in the below figures are potential clock routing options. One goal of these configurations is to have the transmitter clock appear at the cable interface as the Tx Clock signal. This is done either by routing the transmitter clock to the cable Tx Clock signal directly, or by programming the onboard oscillator to the Tx Bit Rate and feeding the oscillator output to the cable Tx Clock signal. When programmable oscillator is the cable Tx Clock source, it is the application's responsibility to program the oscillator to the Tx Bit rate. The following sections present simple examples of four basic cable configuration options. If there is too much disparity between the Tx Bit Rate and the Rx Bit Rate, then data reception and transmission may have be occur on different channels.

4.2.1. Tx Bit Rate == Rx Bit Rate, Cable Rx Clock Used

In this example the receiver's clock comes in over the cable's Rx Clock signal, and the transmitter and the receiver use the same bit rate. This configuration permits the actual transmitter clock to be routed to the cable's Tx Clock signal. This signal routing is illustrated in Figure 3. The code extract below demonstrates the minimum coding for this example configuration. Error checking is omitted for brevity.

```
void config_sample(int fd)
{
    const char*         err     = NULL;
    sio4_hdlc_t         hdlc;
    sio4_hdlc_init_t    init;

   init.tx_bit_rate        = 1000000L;
   init.rx_bit_rate        = 1000000L;
   init.rx_uses_cbl_rxc    = SIO4_HDLC_RX_USES_CBL_RXC_YES;
   init.osc_prog           = 20000000L;
    sio4_hdlc_init(fd, &init, &hdlc, &err);
    sio4_hdlc_set(fd, &hdlc, &err);
}
```

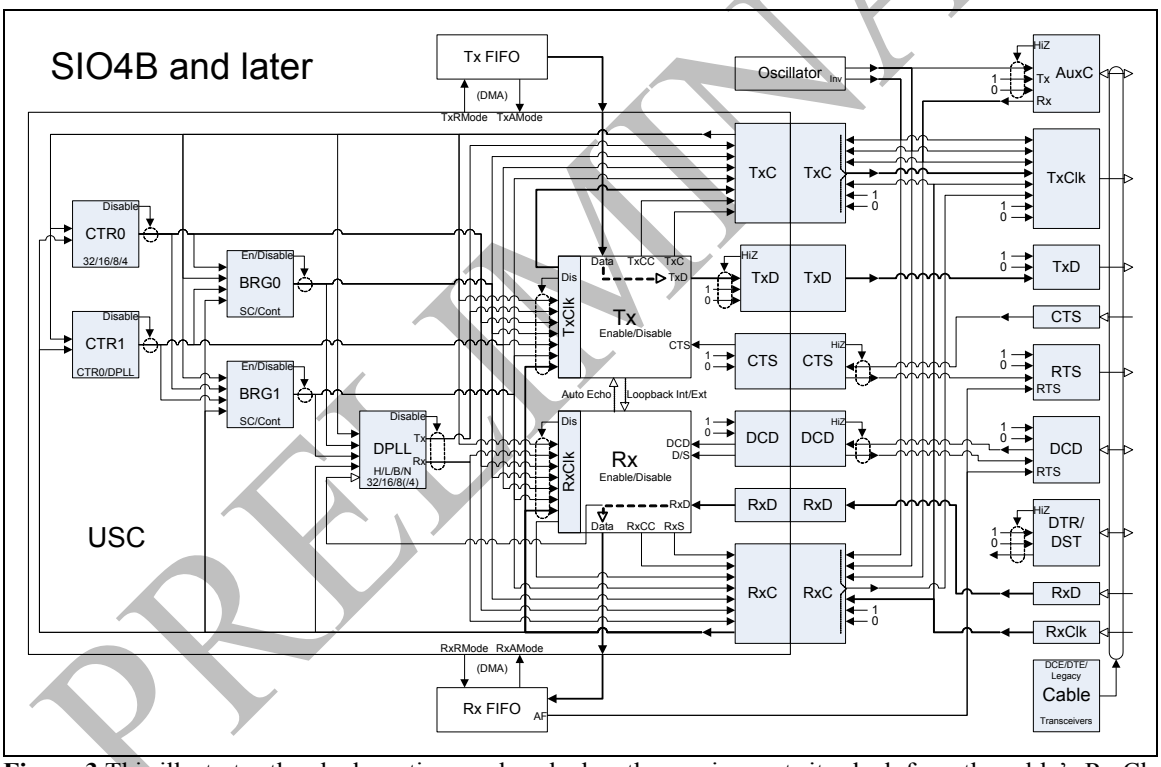

Figure 3 This illustrates the clock routing produced when the receiver gets its clock from the cable's Rx Clock signal, and when the receiver and the transmitter use the same bit rate. In this case the transmitter clock appears at the cable's Tx Clock signal.

4.2.2. Tx Bit Rate != Rx Bit Rate, Cable Rx Clock Used

In this example the receiver's clock comes in over the cable's Rx Clock signal, but the transmitter and the receiver use different bit rates. This configuration does not permit the actual transmitter clock to be routed to the cable's Tx Clock signal. The substitute for this is to program the onboard oscillator to the Tx Bit Rate and route the oscillator output to the cable's Tx Clock signal. This signal routing is illustrated in Figure 4. The code extract below demonstrates the minimum coding for this example configuration. Error checking is omitted for brevity.

```
void config_sample(int fd)
{
    const char*         err     = NULL;
    sio4_hdlc_t         hdlc;
    sio4_hdlc_init_t    init;

    init.tx_bit_rate        = 800000L;
    init.rx_bit_rate        = 1000000L;
    init.rx_uses_cbl_rxc    = SIO4_HDLC_RX_USES_CBL_RXC_YES;
    init.osc_prog           = 800000L;
    sio4_hdlc_init(fd, &init, &hdlc, &err);
    sio4_hdlc_set(fd, &hdlc, &err);
}
```

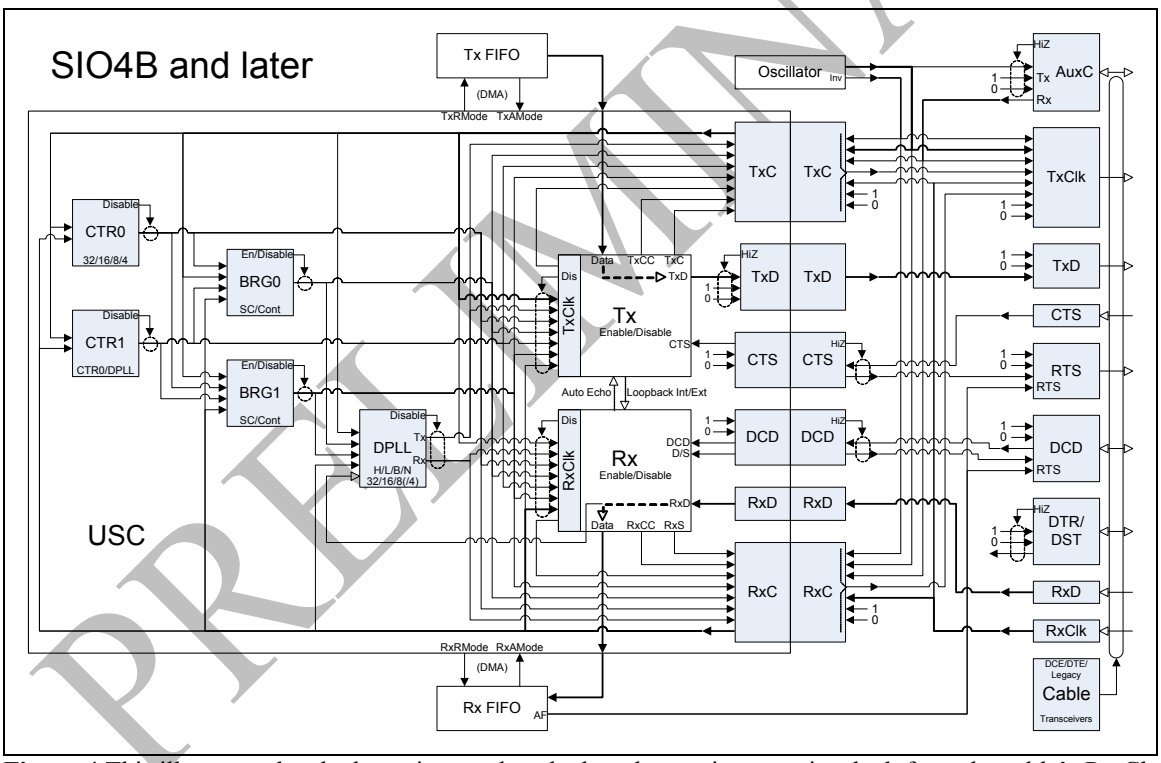

Figure 4 This illustrates the clock routing produced when the receiver gets its clock from the cable's Rx Clock signal, but when the receiver and the transmitter use different bit rates. In this case the oscillator output, which can be programmed to the Tx bit rate, appears at the cable interface as the Tx Clock signal.

4.2.3. Cable Rx Clock Not Used

In this case, the receive clock is derived by the DPLL while the transmit clock is derived from the programmable oscillator. This configuration permits the actual transmitter clock to be routed to the cable's Tx Clock signal. This signal routing is illustrated in Figure 5. 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 programmed oscillator frequency must be chosen so that the USC clocking logic can produce both a matching transmit clock bit rate and a clock rate for the DPLL that is eight, 16 or 32 times the desired receive bit rate. The bit rates produced are reported by the `sio4_hdlc_init()` call. These resulting bit rates should be examined to insure they are suitable.

```
void config_sample(int fd)
{
    const char*         err     = NULL;
    sio4_hdlc_t         hdlc;
    sio4_hdlc_init_t    init;

    init.tx_bit_rate        = 1000000L;
    init.rx_bit_rate        = 1000000L;
    init.rx_uses_cbl_rxc    = SIO4_HDLC_RX_USES_CBL_RXC_NO;
    init.osc_prog           = 20000000L;
    sio4_hdlc_init(fd, &init, &hdlc, &err);
    sio4_hdlc_set(fd, &hdlc, &err);
}
```

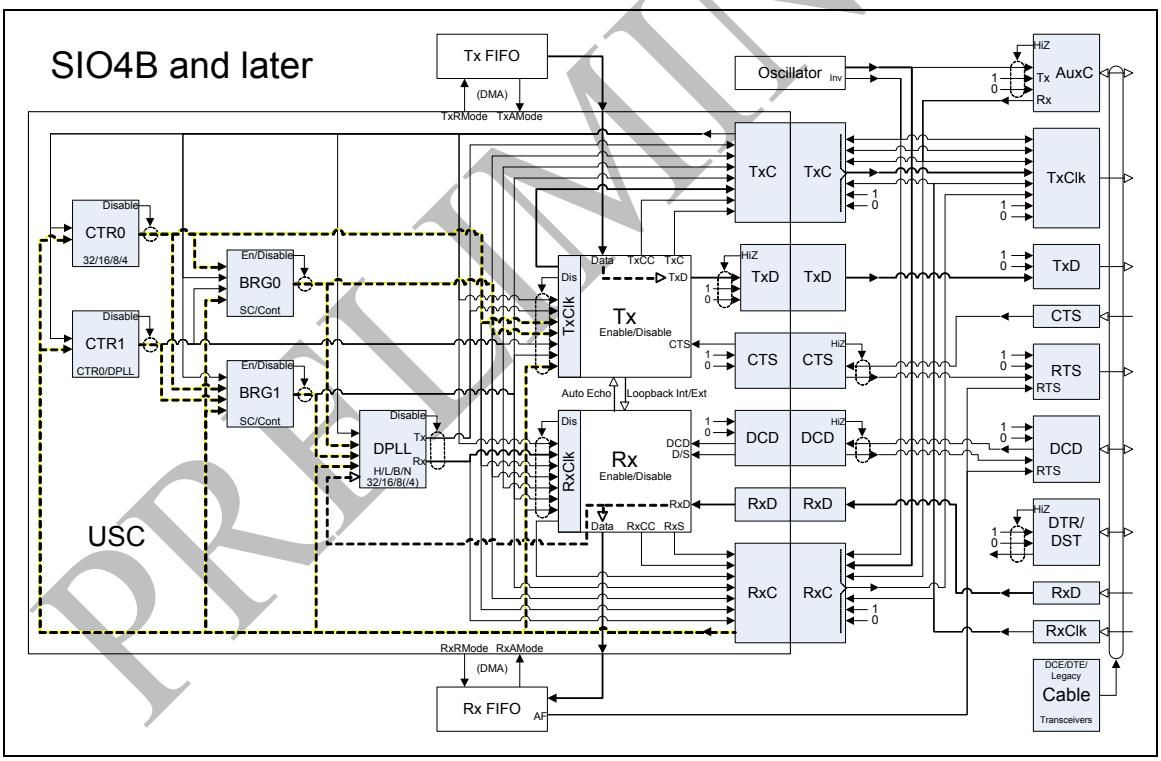

Figure 5 This illustrates the clock routing produced when the receiver derives its clock from the DPLL, and when the receiver and the transmitter use different bit rates. In this case the transmitter clock appears at the cable's Tx Clock signal.

4.3. 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.

4.3.1. 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**

`For (;;)`
`{`
`  …`
`  read SIO4 data`
 	
`  if (error recorded)`
`  {`
`  Error exists in`
`  1) Read buffer, or`
`  2) SIO4 Rx FIFO`
`  Resync data stream.`
`  }`
`  else`
`  {`
`  Read buffer is error free.`
`  }`
`  …`
`}`
	`For (;;)`
	`{`
	`  …`
	`  Enable desired interrupts.`
	`  Wait for an interrupt.`
	
	`  if (error interrupt occurred)`
	`  {`
	`  Record the error.`
	`  }`
	`  …`
	`}`

4.3.2. 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 3.3.1.4, page 33). 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 identity the precise location in the data stream where some Rx related conditions occurr. 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.

4.4. Debugging Aids

The SIO4 driver archive includes two debugging aids appropriate for use with the HDLC Protocol Library. The aids are described below.

4.4.1. sio4_hdlc_show()

The function `sio4_hdlc_show()` (section 3.1.9, page 13) 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 3.3.1, page 20) 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 3.1.8, page 13) or after it is passed to `sio4_hdlc_get()` (section 3.1.7, 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()` calls in order to help explain the results of individual transfer requests.

4.4.2. sio4_reg_list()

The 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()` calls 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 by a call to `sio4_hdlc_open()`.
`gsc`	If non-zero, then the output will include a dump of all `GSC_SIO4_xxx` registers. Refer to `sio4.h` 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 `sio4.h` 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.

4.5. Exclusions

4.5.1. 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 can override the channel specific settings for all four channels. If an application is to access this feature it must be done in parallel with use of the HDLC Protocol Library.

Document History

Revision	Description
September 7, 2015	Updated to library release version 0.9.
December 9, 2014	Updated the current release date.
December 4, 2014	Updated to library release 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 library release 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 library release version 0.6.
October 22, 2013	Updated to library release version 0.5.
October 15, 2013	Updated to library release version 0.4.
August 27, 2013	Initial library release, version 0.2, for the 2.x series driver.