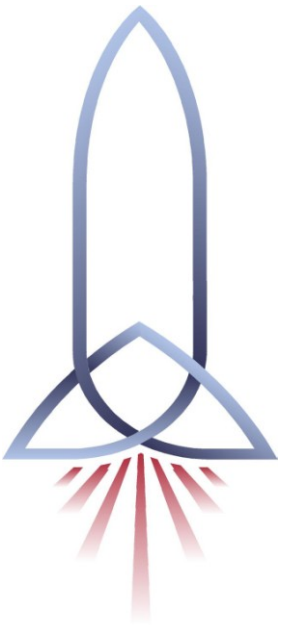

Space
Technology
Centre
University of Dundee

SpaceFibre CODEC

Functional Specification

LIKELY TO CHANGE!

RESTRICTED CIRCULATION ONLY

Prepared by: **Steve Parkes**

Chris McClements

Martin Dunstan

Project Management: **Steve Parkes**

ESA Project Manager: **Martin Suess**

Document Change log

Date	Revision No	Comments	Change Author
31st October 2007	Draft A	Initial	Steve Parkes

Contents

List of Tables

List of Figures

1. INTRODUCTION

1.1 OVERVIEW

This document provides an outline specification for the SpaceFibre CODEC.

This document begins, in Section 2, with an overview of the architecture of the SpaceFibre CODEC and a brief description of how it operates. Section 3 then describes each part of the CODEC in turn. The concepts behind each part are described giving a rationale for the design decisions made. The specification for SpaceFibre is provide in section 4, with section 4.1 providing the interface specification and section 4.2 the functional specification. Section 5 provides an overview of the virtual channel and flow control mechanisms that will be used with the SpaceFibre CODEC.

1.2 TERMS AND DEFINITIONS

1.2.1 List of abbreviations

8B/10B	8-bit to 10-bit
AC	Alternating Current
AD	Applicable Document
ASIC	Application Specific Integrated Circuit
CODEC	Coder/Decoder
CML	Current Mode Logic
D/K	Data/Control
ECSS	European Cooperation for Space Standardization
EDAC	Error Detection and Correction
EEP	Error End of Packet
EM	Electro-Magnetic
EMC	Electro-Magnetic Compatibility
EOF	End Of Frame
EOP	End Of Packet

ESA	European Space Agency
FCT	Flow Control Token
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GAR	Group Adaptive Routing
LSB	Least Significant Bit
LVDS	Low Voltage Differential Signalling
MSB	Most Significant Bit
OS	Ordered Set
PCI	Peripheral Component Interconnect
PLL	Phase-Locked Loop
SEU	Single Event Upset
SOF	Start Of Frame
SOW	Statement of Work
TDA	to be advised
TBC	to be confirmed
TBD	to be defined
UoD	University of Dundee
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

2. SPACEFIBRE CODEC ARCHITECTURAL OVERVIEW

This section provides an overview of the SpaceFibre CODEC architecture and describes how it operates.

2.1 ARCHITECTURE

An overview of the SpaceFibre CODEC architecture is provided in

There are four main functional blocks:

- Virtual Channel and Flow Control: responsible for quality of service channel and flow control over the SpaceFibre link.
- Framing: responsible for framing data to be sent over the SpaceFibre link and for inserting user ordered sets within the data stream.
- Encoding and Link Control: responsible for encoding data into symbols for transmission and vice versa for reception and for initialising the link.
- Serialisation: responsible for serialising and de-serialising SpaceFibre symbols so that they may be transferred over the fibre optic or copper medium.

The latter three blocks form the SpaceFibre CODEC.

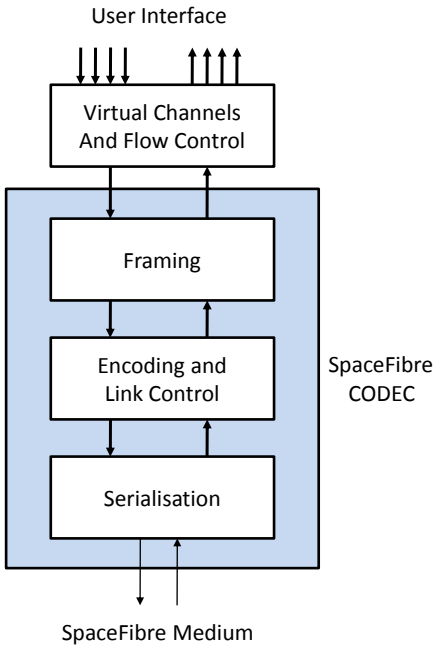

Figure 2-1 Overview of SpaceFibre CODEC

The detailed architecture of the SpaceFibre CODEC is illustrated in Figure 2-2.

The SpaceFibre CODEC is split into several functional layers:

- User interface
- EMC mitigation
- Framing
- Link initialisation and power management
- Data rate adjustment
- 8B/10B encoding and decoding
- Parallel loop-back
- Symbol and ordered set synchronisation
- Serialisation and de-serialisation
- Serial loop-back
- Line driver and receiver

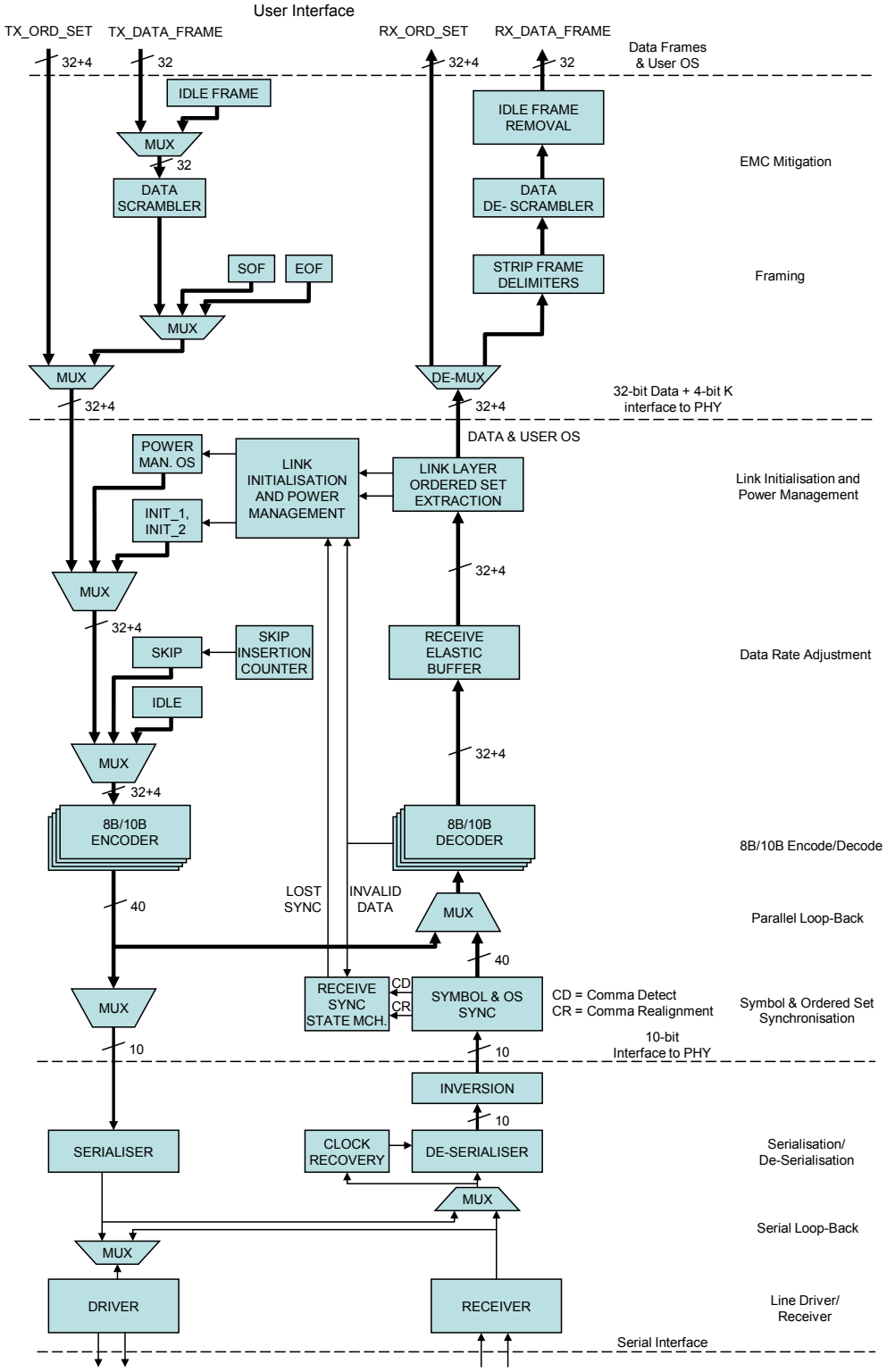

Figure 2-2 SpaceFibre CODEC Architecture

2.1.1 User Interface

The user interface comprises four independent interfaces for sending and receiving data frames and user ordered sets. A user ordered set comprises a four symbol code: starting with a comma and followed by three other symbols either data or K-codes. The user ordered sets will be defined in a higher layer of SpaceFibre and must not conflict with the ordered sets used within the link layer. User order sets support services like time-code and interrupt distribution. A data frame is a series of 32-bit data words (4 bytes) delimited by a start of frame (SOF) and end of frame (EOF). The data frame is limited to a maximum frame size which will be specified by the SpaceFibre standard. Data frames are transferred rather that individual bytes or words of data because it enables handshaking at the user interface to be done much more slowly at the boundaries of the frames rather than on each byte or word boundary.

2.1.2 EMC Mitigation

Sending a constant bit pattern over a serial link can cause high levels of EM emission due to the energy being concentrated in a few frequency components. To avoid this data may be scrambled before transmission by multiplying the data sequence by a pseudo-random sequence. A pseudo-random sequence approximates white noise and thus has a wide bandwidth. Multiplication of the data by the pseudo-random sequence broadens the frequency components, spreading the energy and reducing the peak emission levels.

SpaceFibre uses this technique to mitigate the EM emissions over copper SpaceFibre links. A scrambler is used to scramble the data in each data frame. The original data is then recovered by a de-scrambler at the other end of the link. To help with this process each frame is multiplied by the same pseudo-random sequence i.e. the pseudo-random generator is re-seeded with the same seed at the start of each frame. Ordered sets are not scrambled.

When the link has no other data to send it will send IDLE ordered sets to keep the link active. Sending repeated IDLEs will once again result in excessive EM emission spectral spikes. To avoid this whenever there are no data frames to send an idle frame comprising all zero data is automatically generated. The idle frame is scrambled in the same way as normal data frames, reducing the EM emissions. Idle frames can be terminated, with an EOF, as soon as another data frame becomes ready for transmission, so that the introduction of idle frames does not significantly affect the sending of data frames.

2.1.3 Framing

Framing is the delimiting of data and idle frames by start of frame (SOF) and end of frame (EOF) ordered sets. There are two types of start of frame: start of data frame (SDF) and start of idle frame (SIF), and also two types of end of frame: normal end of frame (EOF) and error end of frame (EEF) which is used when some error is the transmitting system has caused a data frame to be terminated prematurely.

2.1.4 Link initialisation and power management

Link initialisation and power management is handled by a link state machine. In the current study power management is not considered any further.

The link initialisation state machine is responsible for initialising the link prior to transfer of data frames, idle frames or user ordered sets. During link initialisation bit synchronisation, symbol synchronisation and ordered set synchronisation are performed. A handshake protocol is used to ensure that both ends of the link have achieved synchronisation. Speed negotiation is also performed during initialisation.

2.1.5 Data rate adjustment

The two ends of the link operate at the same bit rate within the limits of the crystal oscillators at either end. This is nominally 2.5 Gbits/s ± 100 ppm for a typical crystal oscillator. An elastic-buffer is used in the receiver to compensate for any difference in the clocks at either end of the link. If the transmitter is operating very slightly faster than the receiver then the receive elastic-buffer will start to fill up. If the transmitter is running more slowly than the receiver then the receive elastic-buffer will start to empty. Ideally the receive elastic-buffer should be kept half full. To do this the transmitter sends SKIP ordered sets periodically, at least every 5000 ordered sets. If the receive elastic-buffer is more than half full then the SKIP ordered set is skipped over when reading out of the buffer *i.e.* is ignored and the ordered set or data following is read out instead. This has then effect of reading out two ordered sets rather than one so the buffer becomes emptier. If the receive elastic-buffer is less than half full then the SKIP ordered set is read out but the buffer output pointer is not updated so that the SKIP is read out twice. This slows down the emptying of the receive elastic-buffer and make the buffer fuller.

The receive elastic-buffer is an important element in the SpaceFibre CODEC: it allows data or ordered sets to be read out of the 8B/10B encoder every cycle without having to do a hand-shake to see if a character is ready to be read. This dramatically improves the speed of operation of the interface. Handshaking is done at a higher level, handshaking for complete data frames to reduce handshake overhead.

2.1.6 8B/10B encoding and decoding

The 8B/10B encoding takes an 8-bit input together with a data/control (D/K) flag and provides a 10-bit code for serialisation. The 8B/10B decoder does the inverse operation to recover the 8-bit value and D/K flag. The 8B/10B encoding/decoding is described further in section 3.2.

2.1.7 Symbol and ordered set synchronisation

In the receiver once the bit synchronisation has been achieved and the bit stream recovered (see sections 2.1.8 and 3.4) it is necessary to determine the boundary of the symbols in the bit stream (every 10-bits) and then to identify the ordered set and data set boundary (every 40-bits). The comma symbol (see section 3.2) is used to support both symbol and ordered/data set boundary detection. The comma symbol contains a unique 7-bit code which can be used to detect the boundary between the comma symbol and the next symbol. Ordered sets begin with a comma symbol so the boundary between ordered/data sets can also be detected using the comma symbol.

Ordered sets will be carefully defined so that pairs of ordered sets sent during link initialisation (INITs) and during normal link operation (SKIPs) contain both plus and minus commas. This is to enable the protocol to work with interfaces which only recognise one type of comma.

NOTE: SKIPs and INITs might not have this property yet!

2.1.8 Serialisation and de-serialisation

Serialisation is the conversion of the 10-bit parallel symbols to a bit stream.

De-serialisation is the recovery of a 10-bit parallel stream from a serial bit stream. This requires clock recover from the bit stream using a phase-locked loop (PLL) which is helped by the 8B/10B code giving a transition rich set of transmitted codes.

2.1.9 Line driver and receiver

The serialised data is driven onto a 100 ohm differential impedance line using current-mode logic (CML). This may then be used to drive a fibre optic transmitter. A fibre optic receiver converts the received optical signals back to CML which is then passed over a 100 ohm differential impedance line to the receiver. Both the CML driver and CML receiver are AC coupled to provide galvanic isolation.

CML may be used on its own to provide a copper interface to SpaceFibre.

2.2 TYPICAL OPERATION

The typical operation of the SpaceFibre interface will now be described with reference to Figure 2-2.

Data frames (TX_DATA_FRAME) are passed into the CODEC for transmission. The data is scrambled by the Scrambler to reduce EM emissions from repeating data patterns. If there is no data frame to send an idle frame, containing all zeros, is generated automatically to fill the gap in the data. SOF and EOF are added to the scrambled data to delimit one frame from the next. User ordered sets may be sent at any time taking priority over data being sent. User ordered sets may appear at any point within a data frame.

During link initialisation or link initialisation ordered sets (INIT_1 and INIT_2) are multiplexed into the transmitter to perform the initialisation handshake and configuration under direction of the Link Initialisation state machine. Power management ordered sets are also injected into the data stream at this point but this is outside the scope of the current study.

To support the receive elastic-buffer skip ordered sets are injected into the transmit data stream at a rate determined by the Skip Counter and at an interval no longer than 5000 ordered sets (see section 3.6). Idle ordered sets are sent when the transmitter is in the Active state and there is no other information to send.

The 8B/10B encoder encodes the 8-bit data plus data/control (D/K) flag into a 10-bit code, which has special properties to support data transfer and recovery in the receiver (see section 3.2). The data sets and ordered sets are four symbols wide so four parallel 8B/10B encoders are required to perform the encoding. Alternatively one 8B/10B encoder operating at four times the data/ordered set rate may be used. The 10-bit code from the 8B/10B encoder is passed to the serialiser which converts it into a serial bit stream. This bit stream is driven on to the communications medium by the driver.

The receiver takes the incoming bit stream and recovers the bit stream clock using a phase locked loop. The recovered bit clock is used by the de-serialiser to convert the serial bit stream into a slower speed, parallel data stream which can be handled a little more easily. The 10-bit symbols are separated from the parallel data stream by the symbol and ordered set synchronisation unit. This unit determines the position of comma codes in the 10-bit parallel data stream and then outputs the data correctly aligned on the data set and ordered set boundaries. At this point the data set and ordered sets are 40-bits wide: sets of four 10-bit symbols. The ordered sets all start with a comma symbol..

The 40-bit data is passed to a set of four 8B/10B encoders which decode each 10-bit symbol to 8-bits plus D/K flag. The 8B/10B decoder is run from the received clock. The 8B/10B decoder detects various forms of error: invalid 10-bit code and incorrect running disparity. A receive synchronisation state machine monitors the operation of the Symbol and OS Synchronisation unit and the 8B/10B decoders and determines when the receiver is synchronised or has lost sync.

The set of four decoded symbols are buffered in the receive elastic-buffer, which copes with any differences between the receive clock and the local system clock (see section 3.6).

The output from the receive elastic-buffer is filtered for SKIP, IDLE and Initialisation ordered sets. SKIP and IDLE ordered sets are discarded and initialisation ordered sets are passed to the Link Initialisation state machine. User ordered sets are separated from the data frames and passed to the RX_ORD_SET interface. The SOF and EOF markers are checked and stripped from the data frame. The data is passed up to the de-scrambler, which unravels the scrambled data to reveal the original data. The unscrambled data is then passed to the RX_DATA_FRAME interface. Idle frames are disposed of.

Two loop-back facilities are provided inside the CODEC for test purposes. The first provides loop-back of the parallel 10-bit codes and the second loops-back the serial data before it is driven onto the physical medium. The serial loop-back provides both local and remote loop back, *i.e.* serial data being sent is looped back into the receive de-serialiser and the bit stream data being received is fed back to the line driver.

The Link Initialisation state machine provides support for link initialisation and recovery from errors. During initialisation it ensures that bit and character synchronisations are achieved and that the two ends of the link are both ready to send and receive data.

3. OUTLINE SPECIFICATIONS

This section provides an outline specification for the SpaceFibre CODEC. It considers each function within the CODEC in turn, providing a rationale for the design decisions made and then provides an outline specification.

3.1 DATA SCRAMBLING

Data scrambling is a technique used to reduce the electro-magnetic (EM) emissions from a communications system. The data signal is convolved with a wideband signal which results in the spectrum of the data being broadened. Possible peaks in the EM spectrum of the original data signal are spread out reducing the energy at any single frequency.

A random number generator is used to produce the wideband signal which is XORed with the data being transmitted. At the beginning of each frame being transmitted the random number generator is reseeded with a specific value. A similar random number generator in the receiver, seeded with the same seed as in the transmitter at the start of every new frame, is used to de-scramble the data. The incoming data is XORed with the random number sequence to reveal the original data stream.

The random number generator is implemented using a linear feedback shift register as shown in Figure 3-1. The scrambling/de-scrambling polynomial used is the same as that used in PCI-Express:

$$G(x) = X^{16} + X^5 + X^4 + X^3 + 1$$

The seed for the random number generator is $FFFF_h$ *i.e.* all flip-flops in the random number generator are set to 1.

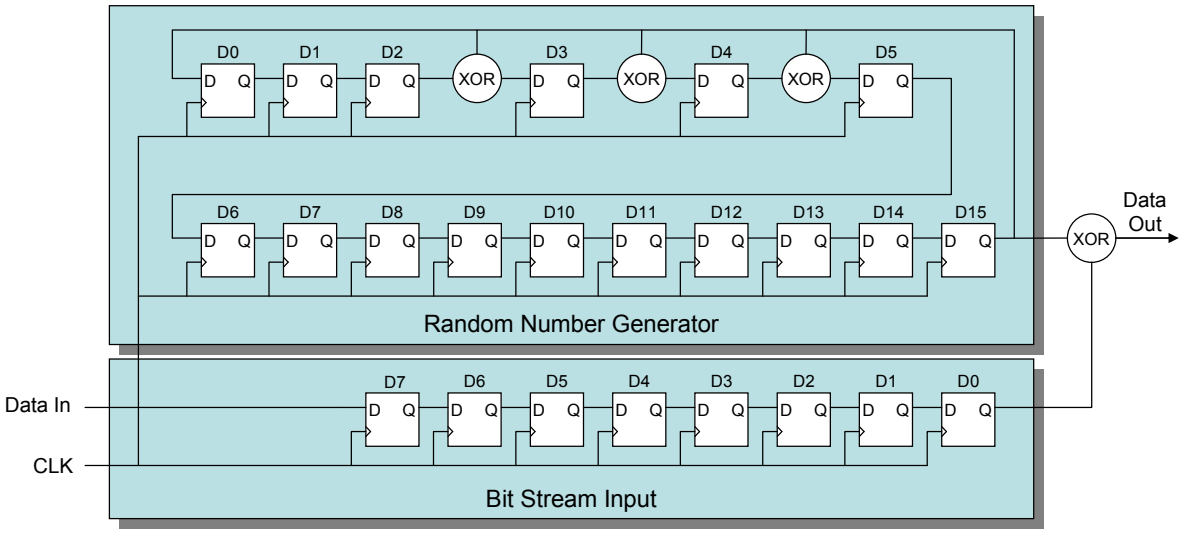

Figure 3-1 Scrambler / De-Scrambler

3.2 8B/10B ENCODING AND DECODING

8B/10B encoding encodes 8-bit data bytes into 10-bit characters for transmission. The 8B/10B encoding has several advantages over direct 8-bit transmission.

1. It provides a transmitted data stream with roughly the same number of 1's as 0's giving the data a zero DC bias, improving the transmission characteristics and enabling AC coupling.
2. Since a 10-bit code has 1024 possible values and not all of these are needed to send an 8-bit value there are spare valid codes left over that can be used for control codes.
3. It guarantees that there will be sufficient number of bit transitions in the serial data stream to enable the recovery of the bit clock using a phase-locked loop. A maximum of five consecutive ones or zeros are ensured with 8B/10B encoding.
4. Since all characters, both data and control characters, are transmitted with 10-bits the bit and character transmission rates are constant simplifying the transmission and reception of characters. This is in contrast to SpaceWire where four different code lengths are possible: 4-bit for control codes, 8-bits for Null, 10-bits for data and 14-bits for time-codes which complicates character transmission and reception.
5. Codes that are unused by the 8B/10B encoding can be used to detect link errors *i.e.* if an unused code occurs then there has been a transmission error.

To avoid significant DC components 8B/10B encoding uses only the 10-bit codes that contain either 5 ones and 5 zeros, 6 ones and 4 zeros, or 4 ones and 6 zeros. There are enough of these to encode the 8-bit data byte and several possible control codes. Characters encoded with 5 ones and 5 zeros

have neutral disparity and will produce zero DC bias. However, if a sequence of bytes was transmitted that contained characters all with 6 ones and 4 zeros the DC component would slowly increase. A similar opposite effect would occur if the characters all contained 4 ones and 6 zeros. To prevent this increasing DC bias and to maintain an equal number of transmitted ones and zeros each character with an unequal number of ones and zeros has two possible codes one with 6 ones and 4 zeros and the other with 4 ones and 6 zeros.

Every time the transmitter sends a character with 6 ones and 4 zeros it will record the fact that it has sent more ones than zeros and the next time it has to send a character with an uneven number of bits it will choose the code that has 4 ones and 6 zeros. This keeps the average number of ones and zeros the same and eliminates any DC bias in the transmitted signal. The Current Running Disparity variable is set to one (positive) when more ones have been sent than zeros and to zero (negative) when more zeros have been sent than ones. Characters with 5 ones and 5 zeros have neutral disparity and do not affect the Current Running Disparity value. When a character with an unequal number of ones and zeros is to be sent, the value of the Current Running Disparity will determine which of the two possible 10-bit codes will be sent. If the Current Running Disparity is positive then the option with 4 ones and 6 zeros is sent, if it is negative then the other option with 6 ones and 4 zeros is transmitted.

Once all 256 possible values of an 8-bit data byte have been assigned a code with 5 ones and 5 zeros or a pair of codes with unequal numbers of ones and zeros, there are just 12 valid codes left out of the possible 1024 values of a 10-bit code. The others have more than six ones or more than six zeros and are invalid.

3.2.1 8B/10B Encoding

8B/10B encoding is normally done using a pair of look-up tables as shown in Figure 3-2 rather than a single look-up table.

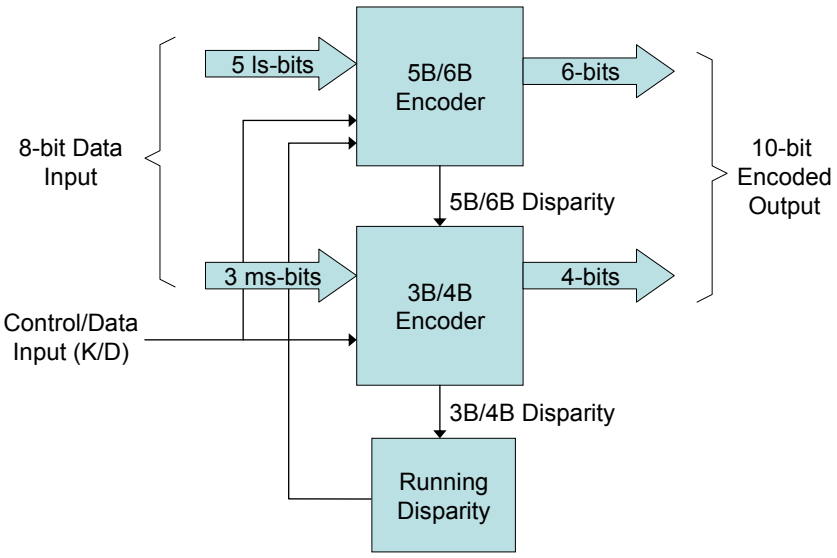

Figure 3-2 8B/10B Encoder

The five least-significant bits are encoded first using a 5B/6B encoder. This takes into account whether the 5 least significant bits are part of a control or data word as determined by the K/D input and also the current running disparity of the link. The 5B/6B encoding table is given in Table 3-1. This table has the following properties:

- The six bit outputs consist of either three ones and three zeros, four ones and two zeros or two ones and four zeros.
- When the output code has neutral disparity (three ones and three zeros) there is one code independent of the running disparity (except for D07.y which has two codes based on the running disparity). The complement of a neutral disparity code also has neutral disparity and, except for D07.y, corresponds to a different input symbol. By definition, using a neutral disparity code will not affect the running disparity.
- When the output code has non-neutral disparity (four ones and two zeros or two ones and four zeros) there are two alternative codes provided which are the complement of each other. The code that is applied when the current running disparity of the link is negative (-ve) has four ones and two zeros which will then make the disparity out of the 5B/6B encoder positive. Similarly when the current running disparity is positive (+ve) the code with two ones and four zeros is applied making the 5B/6B disparity negative.

- The coding table is organised to minimise the amount of logic needed to implement it so that wherever possible there is a one to one mapping of bits from the 5-bit input to the 6-bit output. Note that K28.y must be treated as a special case.

The six-bit output of the 5B/6B encoder forms the six least-significant bits of the 8B/10B encoder output. The 5B/6B disparity is used in the encoding of the three most-significant bits of the 8-bit input data. The 5B/6B disparity, three most significant bits of the input data and the control/data flag (K/D) are fed into a separate 3B/4B encoder which produces the four most significant bits of the 8B/10B encoder output and a new value for the running disparity. The contents of the 3B/4B encoding table are given in Table 3-2. This table has the following properties:

- Only 12 possible codes are valid, those shown in the Table 3-3.
- When the output code has non-neutral disparity there are two codes which are the complement of each other (with the exception of the codes for Dxx.7). One of these two codes will be used depending on the 5B/6B disparity. If the 5B/6B disparity is negative then the option with three ones and one zero will be used resulting in an overall positive disparity which will be the new value of the running disparity. The opposite is the case when the 5B/6B disparity is positive.
- The encoding for Dxx.7 has an alternative coding to prevent five consecutive ones being transmitted. The -ve current running disparity alternative (0111_b) is used for D17.7, D18.7 and D20.7. The +ve current running disparity alternative (1000_b) is used for D11.7, D13.7 and D14.7. This does complicate the encoding somewhat because these special cases have to be identified in the input data stream and the alternative code activated.

The complete 8B/10B encoding is performed by combining the results of the 5B/6B and 3B/4B encoding steps.

3.2.2 8B/10B Decoding

The task of decoding 8B/10B symbols is more complicated than the encoding process since a large number of input codes are mapped to a few valid output codes. Care must be taken to ensure that invalid 8B/10B codes are not accidentally considered to be valid simply because the 5B/6B and 3B/4B components are individually valid. An example is 1110101110_b which has a valid 5B/6B component 111010_b (-D23.y) and a valid 3B/4B component 1110_b (Dxx.7 normal encoding). This is invalid because the alternative Dxx.7 encoding 0111_b encoding ought to have been used.

Additional care must be taken with the 3B/4B decoder since the 4-bit input cannot distinguish between K and D codes. For example, 0110_b represents either -Kxx.1, or Dxx.6 or +Kxx.6.

Table 3-1 5B/6B Encoding			
Input		**Output**	
Data Input	**Data bits 43210 (EDCBA)**	**Current Running Disparity -ve abcdei**	**Current Running Disparity +ve abcdei**
D00.y	00000	100111	011000
D01.y	00001	011101	100010
D02.y	00010	101101	010010
D03.y	00011	110001	
D04.y	00100	110101	001010
D05.y	00101	101001	
D06.y	00110	011001	
D07.y	00111	111000	000111
D08.y	01000	111001	000110
D09.y	01001	100101	
D10.y	01010	010101	
D11.y	01011	110100	
D12.y	01100	001101	
D13.y	01101	101100	
D14.y	01110	011100	
D15.y	01111	010111	101000
D16.y	10000	011011	100100
D17.y	10001	100011	
D18.y	10010	010011	
D19.y	10011	110010	
D20.y	10100	001011	
D21.y	10101	101010	
D22.y	10110	011010	
D/K23.y	10111	111010	000101
D24.y	11000	110011	001100
D25.y	11001	100110	
D26.y	11010	010110	
D/K27.y	11011	110110	001001
D28.y	11100	001110	
K28.y	11100	001111	110000
D/K29.y	11101	101110	010001
D/K30.y	11110	011110	100001
D31.y	11111	101011	010100

Table 3-2 3B/4B Encoding			
Input		**Output**	
Data Input	**Data bits 765 (HGF)**	**5B/6B Disparity -ve fghj**	**5B/6B Disparity +ve fghj**
D/Kxx.0	000	1011	0100
Dxx.1	001	1001	
Kxx.1	001	0110	1001
Dxx.2	010	0101	
Kxx.2	010	1010	0101
D/Kxx.3	011	1100	0011
D/Kxx.4	100	1101	0010
Dxx.5	101	1010	
Kxx.5	101	0101	1010
Dxx.6	110	0110	
Kxx.6	110	1001	0110
Dxx.7	111	1110/0111	0001/1000
Kxx.7	111	0111	1000

The 5B/6B and 3B/4B approach to 8B/10B encoding has lead to a specific notation for representing codes resulting from this encoding. This is illustrated in Figure 3-3.

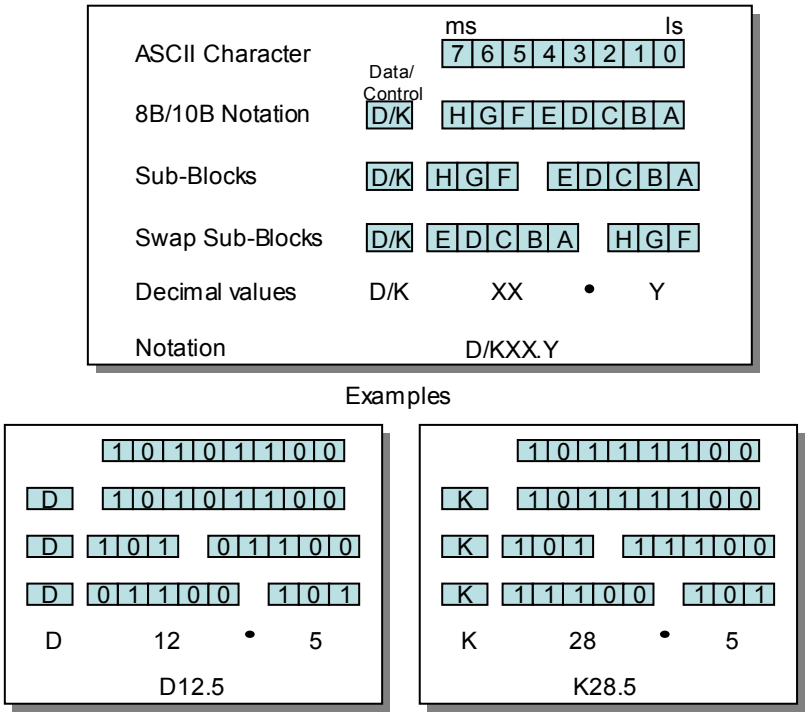

Figure 3-3 8B/10B Notation

The 12 control characters are listed in Table 3-3. Three of these characters (K28.1, K28.5 and K28.7) contain a unique seven bit pattern (0011111 or 1100000) which does not occur in any of the data

codes and which cannot be produced by concatenating any other two data or control codes. This pattern is known as the “comma” pattern and is widely used for performing receive code synchronisation (character alignment). The comma pattern is underlined in Table 3-3.

Note that K28.7 followed by certain other data or control codes can produce a false comma, but the correct one comes first.

Table 3-3 8B/10B Control (K) Codes

Input	Output	
Special Character Name	**Current Running Disparity -ve**	**Current Running Disparity +ve**
K28.0	001111 0100	110000 1011
K28.1	001111 1001	110000 0110
K28.2	001111 0101	110000 1010
K28.3	001111 0011	110000 1100
K28.4	001111 0010	110000 1101
K28.5	001111 1010	110000 0101
K28.6	001111 0110	110000 1001
K28.7	001111 1000	110000 0111
K23.7	111010 1000	000101 0111
K27.7	110110 1000	001001 0111
K29.7	101110 1000	010001 0111
K30.7	011110 1000	100001 0111

The initial disparity can be either positive or negative. The current running disparity of the 8B/10B decoder must always reflect the disparity of the most recently decoded input. This may cause a disparity error to be generated for the first character in a data stream but then the transmitter and receiver will become synchronised. This assumes that the data stream begins with at least one character which has two possible 8B/10B codings based on the current disparity.

Table 3-4 and Table 3-5 show how errors can be captured by monitoring for invalid codes and disparity errors. In Table 3-4 a single bit error converts the D00.0 character sent into a code whose 4B component does not appear in the 3B/4B coding table. This is immediately detected as a coding error.

Table 3-4 Detection of error by invalid code								
Character	Transmitted				Received			
D00.0	100111	-ve	0100	-ve	100111	-ve	0**0**00	invalid

In Table 3-5 an error occurs in the first line with the D08.1 character being changed to the D05.1 character. D05.1 is a valid character so goes undetected. The running disparity should however be positive but because of the error it is negative. The characters that follow have neutral disparity so the running disparity remains unchanged and no error is detected. Eventually a character, D15.1, is sent which does not have neutral disparity. At the transmitter the running disparity is negative prior to D15.1 so that character is encoded as 101000 1001 which has negative disparity. When this is received at the receiver the negative disparity causes an error because the running disparity there is already negative. The error has been caught by disparity but several characters were sent before the error became apparent.

The receiver should look out for both invalid characters and disparity errors. It is also important that a CRC code is added to each packet sent to ensure that any error in a packet is detected.

Table 3-5 Detection of error by invalid disparity									
Character	Transmitted				Received				Character
D08.1	111001	+ve	1001	+ve	1**0**1001	-ve	1001	-ve	D05.1
D09.1	100101	+ve	1001	+ve	100101	-ve	1001	-ve	D09.1
D10.1	010101	+ve	1001	+ve	010101	-ve	1001	-ve	D10.1
D11.1	110100	+ve	1001	+ve	110100	-ve	1001	-ve	D11.1
D12.1	001101	+ve	1001	+ve	001101	-ve	1001	-ve	D12.1
D13.1	101100	+ve	1001	+ve	101100	-ve	1001	-ve	D13.1
D14.1	011100	+ve	1001	+ve	011100	-ve	1001	-ve	D14.1
D15.1	101000	-ve	1001	-ve	101000	ERROR			D15.1

3.3 SERIALISATION AND DE-SERIALISATION

Serialisation is the conversion of a parallel data stream into a serial one. This is straightforward. The parallel 10-bit data word is loaded into a shift register and then shifted out using a transmit clock signal to drive the shift register. A new character has to be loaded into the parallel input of the shift register as soon as the previous 10-bit character has been shifted out to prevent a gap in the serial data.

De-serialisation is the opposite of serialisation. The serial data is shifted into a shift register using a receive clock (also called a bit clock). Recovery of the receive clock from the transmitted serial data stream is described in section 3.4 Once a full 10-bit character has been shifted into the shift register it is read out in parallel. The 10-bit character must be read out at the correct point in the serial data stream *i.e.* when a complete new 10-bit character is in the shift register. Character synchronisation is described in section 3.5.

3.4 RECEIVE CLOCK RECOVERY

Recovery of the receive clock (bit clock) from the received serial data stream is done using a phase-locked loop (PLL). A typical phase-locked loop is shown in Figure 3-4

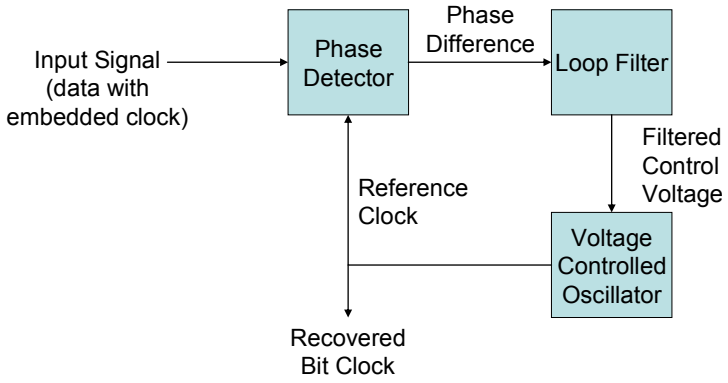

Figure 3-4 Typical Phase-Locked Loop

The phase of the incoming data stream is compared to the phase of a reference clock signal. The detected phase difference is filtered, removing noise and providing an average phase difference. The filtered phase difference is used to control the frequency of the reference clock. If there is a positive phase difference with edges in the data stream occurring before edges in the reference clock then the reference clock frequency must be increased so that it catches up with the data stream edges. If there is a negative phase difference then the reference clock is occurring too early so must be slowed down, in which case the reference clock frequency is reduced.

When locked so that there is no phase difference, the reference clock can be used to recover the data-bits from the serial stream.

High frequency phase locked loops are normally implemented using a voltage controlled oscillator and an analogue loop filter. Since the PLL is analogue it will not be defined in VHDL.

The time taken for a PLL to lock onto a signal is dependent upon the design of the PLL and the difference between the input bit stream phase and the PLL reference clock phase. Typically it takes at least 5000 edges in the bit stream for a PLL to lock although it can be substantially longer for some PLL designs.

3.5 CHARACTER SYNCHRONISATION

Character synchronisation is necessary in the receiver to separate out each character from the received bit stream. To do this is it necessary to identify where a character starts, after that each individual character can be separated by simply counting 10-bits for each character. Identifying the start of a character is used using the 8B/10B Comma bit sequences. Comma sequences are unique seven bit sequences:

- Plus Comma 0011111
- Negative Comma 1100000

An illustration of character synchronisation using a plus comma is shown in Figure 3-5. The start of the next character occurs on the fourth bit after the end of the detected plus comma.

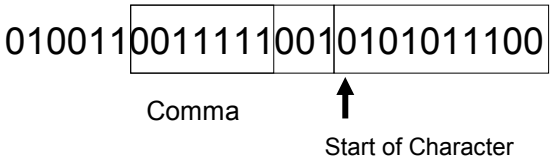

Figure 3-5 Character Synchronisation Using a Plus Comma

There are two principal means of performing character synchronisation. The first method, shown in Figure 3-6, performs the character synchronisation after de-serialisation, while the second method, illustrated in Figure 3-7, does it during de-serialisation.

Figure 3-6 shows the received bit stream being fed into the de-serialising shift register. As soon as ten bits have been received the de-serialised data is loaded into a register. The exact position of the ten bits in the data stream is not important. After a further ten bits have been received the data in the register is loaded into a second register and the de-serialised data is loaded into the first register. The

20 bits in these two registers are examined for a possible comma, using the comma detect circuitry. The combinatorial logic in the comma detect circuitry outputs a Comma Detect signal when a comma is found and registers the position of the start of the next character. The Start of Character is used to drive a 20:10 multiplexer which select the ten-bits of a character from the 20-bits in the two registers.

With this approach comma detection is done at a clock rate of one tenth of that of the bit stream. The comma detect circuitry has to simultaneously look for a comma in ten possible bit positions, requiring 10 correlators.

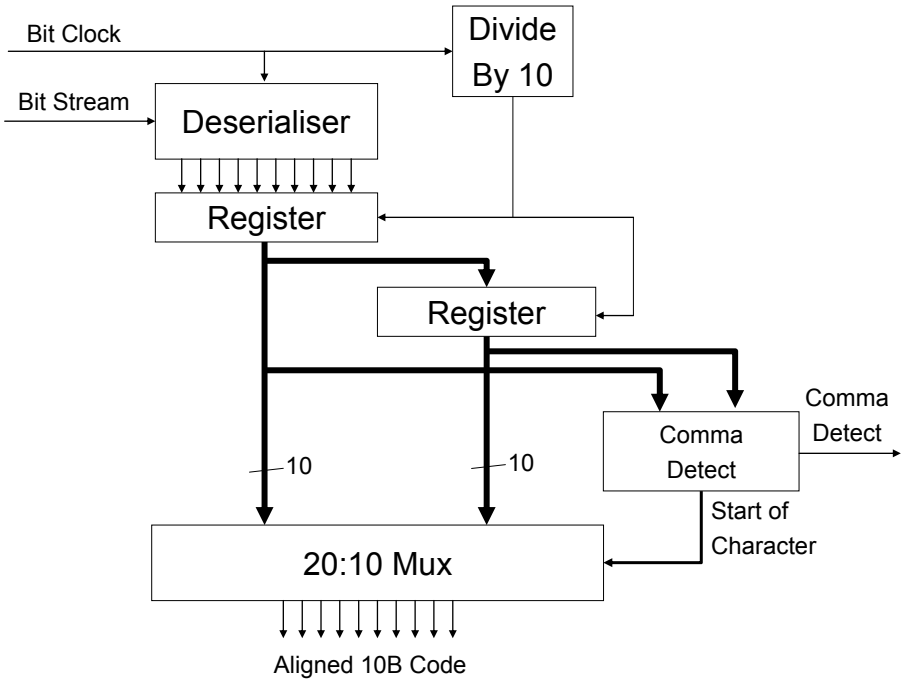

Figure 3-6 Character Alignment After De-serialisation

The other approach, performs comma detection at the bit clock rate. The bit stream is fed into a 10-bit shift register (de-serialiser). The 10-bit parallel output from the shift register fed to a 10-bit character register and to a Comma Detect circuit. The Comma Detect circuit looks for a Comma in the last seven bits of the shift register (*i.e.* the first seven bits to enter the shift register). When a comma is detected the data in the shift register are loaded into the data register. A bit counter is used to count the 10-bits in each character, loading the data register from the shift register every 10 bits. The comma detect circuit resets the counter, re-synchronising the bit counter and forcing the data in the shift register to be loaded into the data register.

This approach requires the comma detect circuitry to operate at the rate of the bit clock and needs a high-speed bit counter. The amount of circuitry is significantly less that the other approach.

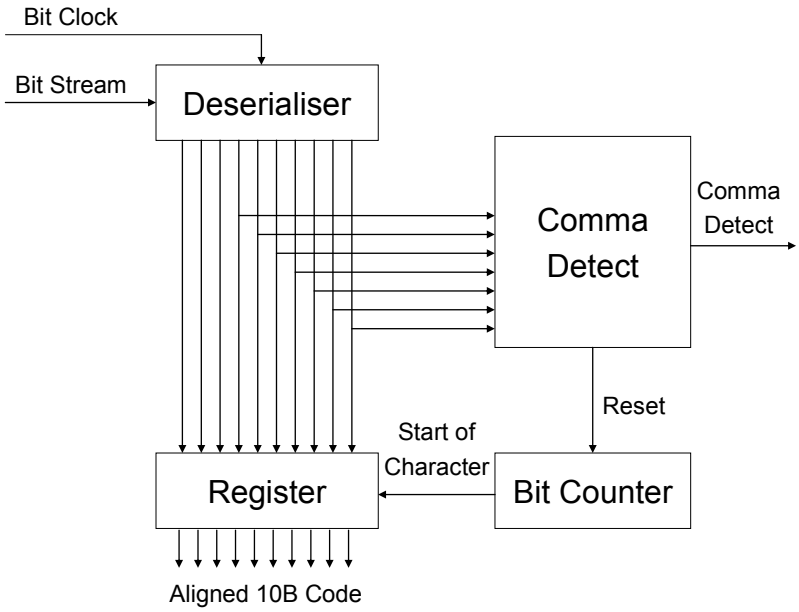

Figure 3-7 Character Alignment During De-serialisation

3.6 RECEIVE ELASTIC BUFFER

The two ends of a link are both expected to operate at the same frequency. In practice, however, there will be slight differences in the clocks at the two ends of the link. This can cause receive buffer overflow or under-run problems unless the difference in the two clock speeds is compensated for. This is achieved using a Receive Elastic Buffer and associated SKIP characters.

The receive clock (bit clock) is recovered from the incoming bit stream, so is at the same frequency as the transmit clock at the other end of the link. After de-serialisation and character synchronisation the incoming data must be transferred from the receive clock domain to the local system clock domain. In passing between these two clock domains, slight differences in the clock frequencies must be accommodated. This is achieved using the Receive Elastic Buffer.

The normal situation with a Receive Elastic Buffer is illustrated in Figure 3-8. Data is written into the buffer using a write pointer which operates at the receive character rate (RXRECCLK). It is read out by read pointer which operates at the user system character rate (RXUSRCLK).

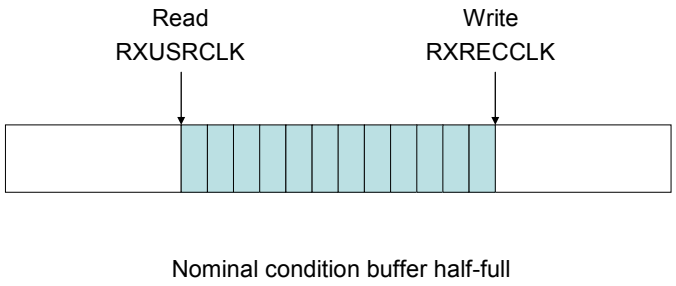

Figure 3-8 Receive Elastic Buffer - Nominal Condition

Any difference between the two clock frequencies is compensated for using a special character, SKIP, inserted every so often into the data stream

If the RXUSRCLK is faster than RXRECCLK the buffer will slowly empty. When the buffer is less than half full, implying that RXUSRCLK is faster than RXRECCLK, extra SKIP characters are added to the Receive Elastic Buffer. This may be done when a SKIP character is read out of the buffer, by simply not incrementing the read pointer, so that the SKIP character will be read a second time. The effect is to add an extra SKIP character to the data stream, temporarily slowing down the RXUSRCLK to compensate for it being faster than RXRECCLK. This is illustrated in Figure 3-9.

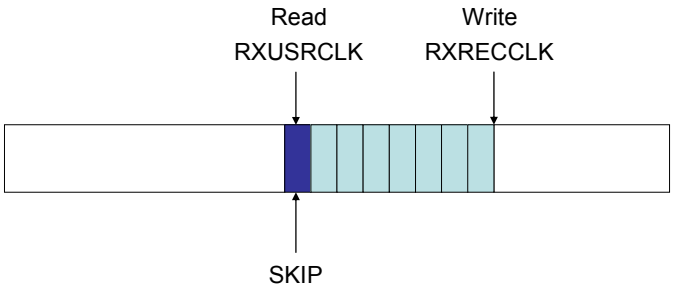

Figure 3-9 Receive Elastic Buffer Emptying

If RXUSRCLK is slower than RXRECCLK then the Receive Elastic Buffer will slowly fill up. When buffer more than half full, SKIP characters are skipped. This is done by incrementing the read pointer past a SKIP character, *i.e.* if after reading a character, the next character to be read is a SKIP character, it is ignored and the read pointer is moved to point to the following character instead. The effect is to remove SKIP characters from the buffer, temporarily speeding up the RXUSRCLK to make up for the fact that it is slower than RXRECCLK. This is shown in Figure 3-10. Note that the SKIP operation requires the elastic buffer to know in advance that a SKIP is present in the buffer without reading it otherwise the SKIP operation will have no effect.

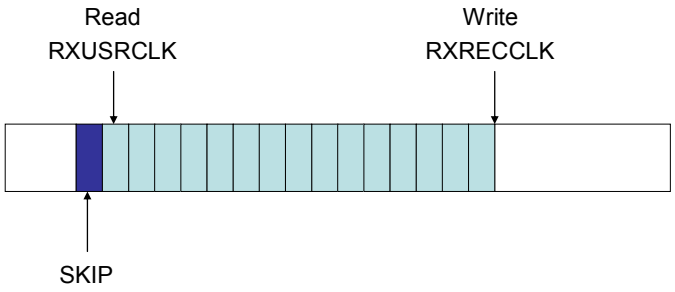

Figure 3-10 Receive Elastic Buffer Filling Up

For the Receive Elastic Buffer to work properly there must be sufficient SKIPs in the data stream, so that they can be remove if necessary. The frequency of SKIPs depends on the size of the elastic buffer and the maximum frequency difference between RXUSRCLK and RXRECCLK.

Assume that the nominal operational frequency is F characters per second and that the maximum clock difference, is D Hz, then the time, T, taken for the elastic buffer to have one character too many or one character too few is given by:

T = 1 / (Receive Clock Frequency – User Clock Frequency)

T = 1/ ((F+D) – (F-D))= 1/ (2D)

In this time the number of characters sent is

N = (F+D).T

which is approximately

N ≈ F/(2D)

since F is much greater than D.

Now D/F is the maximum clock drift, so

N ≈ 1/(2P)

Where P is the maximum drift in the clock.

For a ±100 ppm maximum clock drift, which is readily achievable using crystal oscillators, D, is 10^{-4}, and the number of characters sent before the elastic buffer is one character out is 5000. A SKIP character must thus be sent every 5000 characters to prevent the Elastic buffer from ever being more than one character out. This is the case independent of the size of the characters.

SKIPs will be inserted in pairs and they will be defined such that the comma at the start of one SKIP will be a positive comma while the one at the start of the other skip will be a negative comma. This is to allow devices that recognise only one type of comma to function correctly.

3.7 RECEIVE SYNCHRONISATION STATE MACHINE

The Receive Synchronisation state machine is used to determine and indicate when incoming data is properly synchronised at the bit and symbol level.

Bit synchronisation occurs once the receive clock recovery circuit has locked on to the bit transitions in the receive serial bit stream. If receive clock recovery is done with a PLL then bit synchronisation occurs when the PLL is locked or has been locked for a specified period of time.

Loss of bit synchronisation occurs when the receive clock recovery circuit losses lock to the bit transitions in the receive serial bit stream.

Symbol synchronisation occurs once symbols can be distinguished in the received serial data. Symbol boundaries can be identified using the unique comma sequence. One a comma code has been detected symbol synchronisation has occurred. To avoid incorrect symbol synchronisation due to noise it is prudent to confirm that valid symbols are being received after a comma sequence has been received. If this is that case then the receiver is synchronised and ready.

A loss of symbol synchronisation occurs when several invalid symbols have been received. A single symbol in error may mean that a transitory error occurred without losing synchronisation. A counter is used to count the number of invalid symbols. This counter is incremented by INVALID_INCR (e.g. four) every time an invalid symbol is received, and is decremented by one every time a valid symbol is received. Assuming that just one invalid symbol is received, followed by many valid symbols, *i.e.* there is a transitory error, the INVALID_INCR value determines how many valid symbols must be received before the arrival of the invalid symbol is completely forgotten. For example, if RX_LOS_INVALID_INCR is four then when an invalid symbol arrives, the counter is incremented to four. Each subsequent valid symbol decrements this counter by one until it reaches zero, at which point the invalid symbol has been forgotten. If several invalid symbols arrive then the counter may reach a LOS_THRESHOLD value in which case loss of synchronisation is triggered. Continuing the previous example, if LOS_THRESHOLD is set to 12 it will take 3 successive invalid symbols to trigger loss of synchronisation, or 4 invalid symbols with one valid symbol between each of them.

The Receive Synchronisation state machine is illustrated in Figure 3-11.

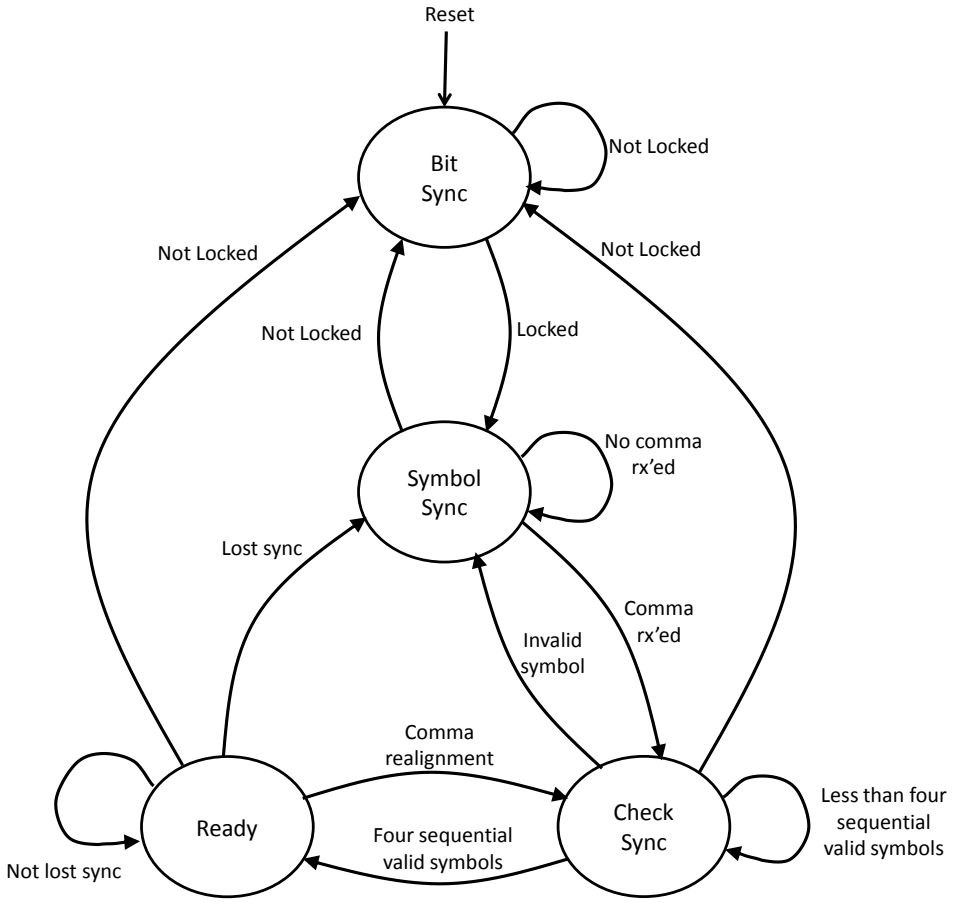

Figure 3-11 Receive Synchronisation State Machine

There are four states:

- BitSync – the receiver is waiting for bit synchronisation to be achieved:
  - Waits until the receive clock recovery circuit indicates that bit synchronisation has been achieved and then moves to the SymbolSync state.
- SymbolSync – the receiver is in the process of symbol synchronisation:
  - Waits for a Comma to be detected.
  - When a Comma is received the next state is CheckSync.
  - If the receive clock recovery circuitry indicates that bit synchronisation has been lost then the next state is BitSync.
- CheckSync – the receiver is checking that proper symbol synchronisation has been achieved:
  - Waits for four consecutive valid symbols.

- When four consecutive valid symbols have been received synchronisation has been acquired and the state machine moves to the Ready state.
- If an invalid symbol is received then symbol synchronisation has not been acquired so the state machine moves back to the SymbolSync state.
- If the receive clock recovery circuitry indicates that bit synchronisation has been lost then the next state is BitSync.

- Ready – the receiver is synchronised:
  - The counter is decremented by one for each valid symbol received.
  - The counter cannot be decremented below zero.
  - The counter is incremented by INVALID_INCR for each invalid symbol.
  - If the counter exceeds the LOS_THRESHOLD value then the next state is the SymbolSync state.
  - If a Comma realignment is detected it implies a change in the character synchronisation. resynchronisation is necessary so the next state is CheckSync.
  - If the receive clock recovery circuitry indicates that bit synchronisation has been lost then the next state is BitSync.

3.8 LINK STATE MACHINE

The link state machine is responsible for getting the link up and running and for recovering from any errors. The same state machine will also be responsible for power management and for unidirectional link operation but this is beyond the scope of the current study.

The Link state machine is illustrated in Figure 3-12.

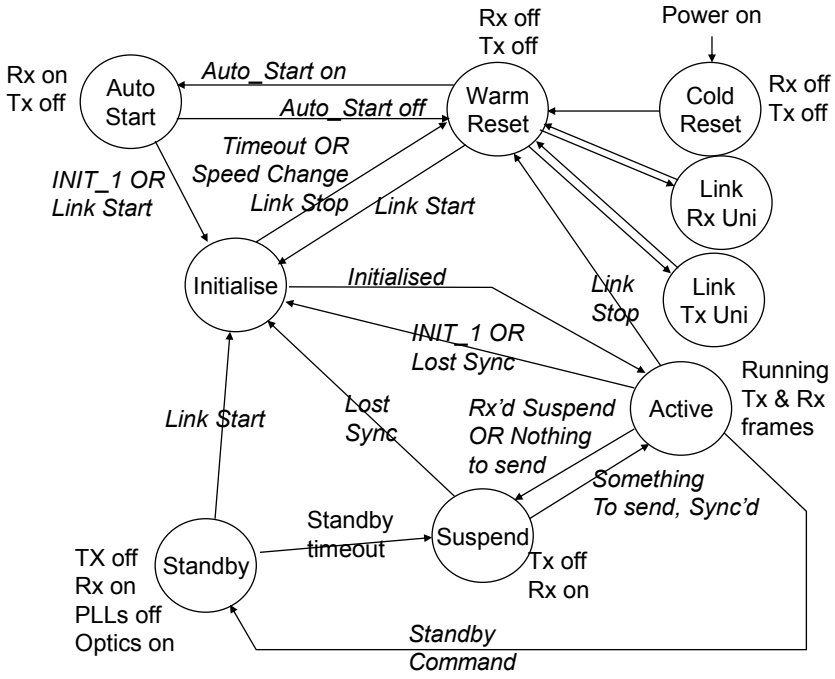

Figure 3-12 Link State Machine

There are five principal states: cold reset, warm reset, auto-start, initialise and active. In addition there are two, or possibly more, power states: suspend and standby, and two states used to support unidirectional link operation: Link Tx Uni and Link Rx Uni. Only the five principal states are considered in this document.

The cold reset state is entered on powering up the link interface. In the cold reset state all link temporary registers and configuration registers are reset to their default values. After cold reset is complete the initialisation state machine moves to the warm reset state.

In the warm reset state the link temporary registers are reset to their default values. The link waits for a command from the local user application (host system). When an auto-start or link start command is give, by setting the auto-start bit or link start bit in a configuration register, the link state machine moves to the AutoStart state or Initialise state, respectively. In both the cold reset and the warm reset state both the receiver and transmitter are disabled.

In the AutoStart state the receiver is enabled to receive data. In this state the link state machine is waiting to receive data. When data is received and the receiver locks on to the received bit and symbol stream the link state machine moves to the Link Start state. If the link start command (link start bit set) is given then the link state machine moves immediately to the Link Start state without waiting

for data to be received first. If incorrect polarity of the input stream is detected then the polarity of the receiver may be inverted once.

In the Initialisation state the transmitter starts to send and listen for INIT_1 and INIT_2 ordered sets as part of the initialisation handshake. This is described later in Section 3.9. When the initialisation handshake is complete the link state machine moves to the Active state. If there is an initialisation timeout or the link is stopped (Start and Auto Start off) while in the Initialisation state then the link state machine moves back to the warm reset state. If incorrect polarity of the input stream is detected then the polarity of the receiver may be inverted once.

In the Active state the link can send and receive user ordered sets and data frames. If there is a persistent error and synchronisation is lost then the link state machine returns to the Initialisation state. If it receives an INIT_1 ordered set while in the Active state this indicates that the other end of the link has lost synchronisation: the state machine moves to the Initialisation state and the link is re-initialised to attempt to recover from the error. If the link is stopped (Link Start and Auto Start are both disabled) then the link state machine moves to the warm reset state.

Note that a receiver polarity correction may be made at most once after leaving warm reset until the state machine returns to warm reset (directly or indirectly via a cold reset).

3.9 LINK INITIALISATION

The Link Initialisation state comprises several sub-states which are illustrated in Figure 3-13.

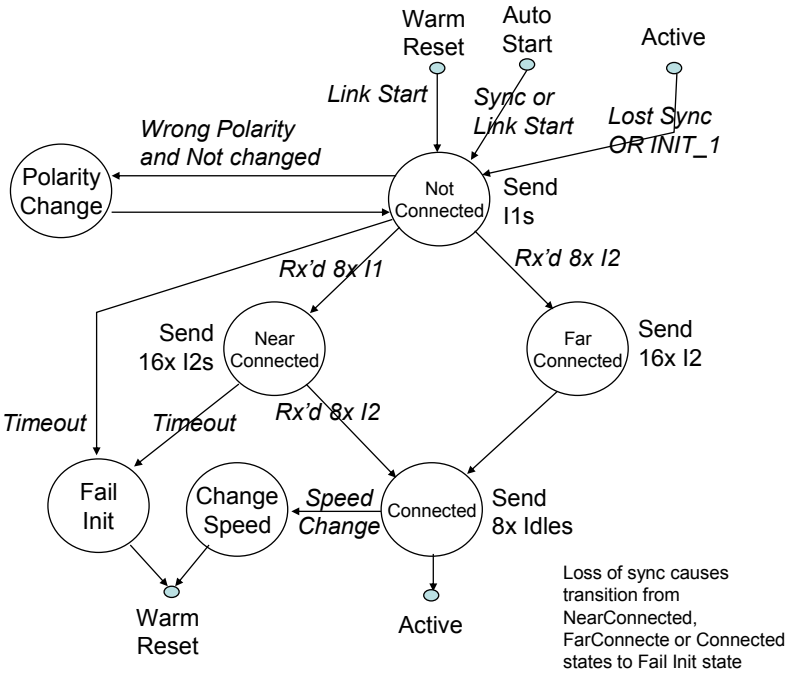

Figure 3-13 Link Initialisation State Machine

The Link Initialisation state machine achieves a connection using a simple bi-directional handshake. INIT_1 ordered sets are transmitted by the near end. When the far end has synchronised and received eight consecutive INIT_1s it acknowledges this by sending INIT_2. Receiving INIT_1s allows the link interface to synchronise. Receiving INIT_2s indicates that the far end has synchronised.

There are four main states in the Link Initialisation state machine which are named according to the knowledge about the near end and far end of the link. If the near end has synchronised properly and received some INIT_1 initialisation ordered sets then it can be considered as being connected, prior to this it is not connected. The receipt of INIT_2 initialisation ordered sets indicates that the far end has received initialisation ordered sets, synchronised and become connected. The four primary states are:

- NotConnected: Near end not connected, far end not connected. This is the situation before either end has synchronised and become connected.
- NearConnected: Near end connected, far end not connected. This is the situation when the near end has synchronised on the incoming INIT_1 ordered sets and received 8 consecutive INIT_1s.
- FarConnected: Near end not connected, far end connected. The far end has synchronised first, before the near end. Having achieved synchronisation the far end has started to send INIT_2s which have been received at the near end. In fact both ends are synchronised but the near end synchronisation has to be signalled by sending INIT_2s to the far end.
- Connected: Near end connected, far end connected. Both the near end and far end have synchronised successfully so the link is ready to transfer data.

The NotConnected state is entered from the warm reset state when the link start bit is set, from the auto-start state when an INIT_1 has been received or when the link start bit is set, or from the Active state when receiver synchronisation is lost (lost_sync) or an INIT_1 ordered set is received, indicating that the receiver at the far end of the link has lost synchronisation. In this state the link interface transmits INIT_1 ordered sets so that the far end of the link can synchronise. If the far end of the link is also in the NotConnected state then it will also be sending INIT_1s they will be received by the near end receiver causing it to synchronise and star receiving the INIT_1 ordered sets. When the receiver has received eight consecutive INIT_1s the link initialisation state machine moves to the NearConnected state. If it receives eight consecutive INIT_2s before it receives eight consecutive INIT_1s then the far end has made a connection before the near end so the state machine moves to the FarConnected state. A timeout timer is started on entering the NotConnected state, if this timeout timer expires then the link has failed to initialise and the state machine moves to the Fail Init state.

The NearConnected state is entered from the NotConnected state when the receiver has received eight consecutive INIT_1s. This means that the near end has correctly synchronised. It signals this condition back to the far end of the link by sending INIT_2s. IN the NearConnected state when eight consecutive INIT_2s have been received it means that the far end has synchronised successfully so the link initialisation state machine moves to the Connected state. A timeout timer is started on entering the NearConnected state, if this timeout timer expires then the link has failed to initialise and the state machine moves to the Fail Init state.

The FarConnected state is entered from the NotConnected state when the receiver has received eight consecutive INIT_2s. This indicates that the far end of the link has initialised quickly before the near end has synchronised and received eight INIT_1s. In the FarConnected state communication in both directions has been established. It only remains to signal the fact that the near end has synchronised to the far end. It does this by sending 16 INIT_2s on the assumption that since both ends of the link are synchronised eight of these INIT_2s will be received consecutively by the far end. As soon as the 16 INIT_2s have been sent the link initialisation state machine moves unconditionally to the Connected state.

In the Connected state the link is synchronised in both directions. To allow both ends to finish sending any outstanding INIT_2s eight IDLE ordered sets are transmitted and then the link state machine moves to the Active state. In the Connected state if a speed change is required then the link initialisation state machine moves to the Speed Change state rather than the Active state.

In addition to the four states primary states there are three secondary states:

- Polarity Change: This state is entered if the polarity of the receiver input is found to be inverted. The polarity of the receiver is inverted to compensate for this and the state machine reverts back to the NotConnected state to complete initialisation. The polarity may not be changed more than once without passing through warm reset. Note that if the polarity was changed while in auto-start then it may not be changed again when in any of these states unless the warm-reset state has been visited first.
- Speed Change: The INIT_1 and INIT_2 ordered sets contain information about the maximum operating rate of the end of the link that sent those ordered sets. The speed change state is entered after initialisation is complete if it is determined that both end of the link are capable of operating at a higher rate than the current operating rate. The link speed is set to the maximum rate that both ends of the link can operate at and then the link state machine goes back to the warm reset and re-initialises at the new data rate.
- Fail Init: If the link fails to initialise then this state is entered. The cause of the failure can be recorded in a set of persistent registers (*i.e.* registers not cleared by warm reset). This information can then be used by higher layer software to determine what to do in the event o a

persistent failure. For example if the link continually fails to initialise at the highest common speed the link may be initialised at a lower operating speed.

3.10 LINK INITIALISATION SEQUENCE DIAGRAMS

In this section several sequence diagrams are presented showing the operation of the link initialisation state machine under various conditions.

3.10.1 Initialisation from AutoStart

Operation of the Link Initialisation state machine when one end (End B) is in the Auto-Start state and the other end (End A) has its link interface started (Link Start) is illustrated in the sequence diagram of Figure 3-14. In the sequence diagrams used in this document time goes from the top of the diagram to the bottom and the lines from top to bottom represent the life of a state machine. The narrow rectangles drawn on a state machine represent each state of the state machine with the name of the state being written adjacent to each rectangle (*e.g.* NotConnected). The events that cause transitions from one state to another are written in italics (*e.g. Rx'd 8x Init 1*). Arrows between the two link end state machines represent the transfer of relevant information between the two state machines, *i.e.* across the link, (*e.g.* Init 1). Arrows coming from the left-hand side of the diagram represent external events from the user application (host system) at End A (*e.g. Link Start*), while those from the right-hand side are from the user application at End B (*e.g. Auto_Start*).

In Figure 3-14, End B is given an auto_start command putting it into the Auto_Start state, where it waits for a bit stream to start to come in on its receiver. End A is given a link start command and moves to the NotConnected state, where it starts sending INIT_1s. These INIT_1s are received at End B which synchronises to the bit stream and moves to the NotConnected state. End B now starts sending INIT_1s. At least 16 INIT_1s must be sent. These INIT_1s are received at End A causing the receiver to synchronise. End B receives eight INIT_1s and moves to the NearConnected state where it starts sending INIT_2s to indicate to End A that it is synchronised. End A in the meantime has synchronised to the bit stream and received eight INIT_1s. Both ends now start sending INIT_2s. When each end has received eight INIT_2s it moves to the Connected state, sends eight IDLEs and then moves to the Active state where it can send and receive data.

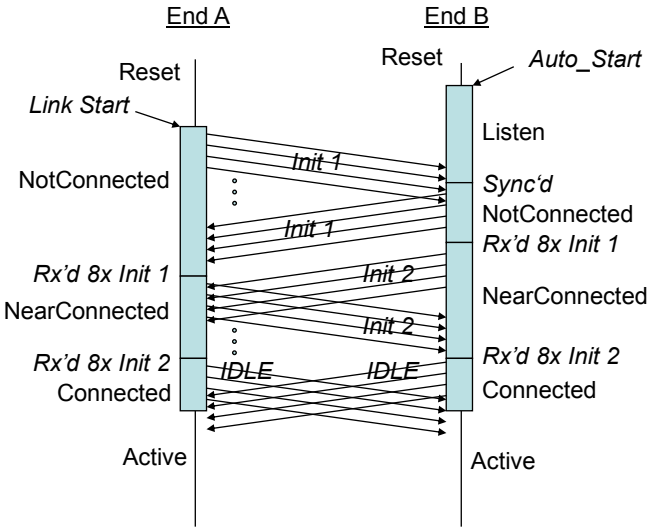

Figure 3-14: Initialisation from Auto-Start

The situation where End A does not receive eight INIT_1s before it starts to receive INIT_2s is illustrated in Figure 3-15. End B has been auto_started and is listening for data arriving at its receiver. End A is instructed to link start and begins sending INIT_1s. These arrive at End B causing the receiver to synchronise, the state machine to move to NotConnected and the transmitter to start sending INIT_1s to End A. It is possible now for End B to immediately receive eight consecutive INIT_1s since its receiver is synchronised. It is likely to do this before End A has synchronised on the stream of INIT_1s being sent by End B. End B moves to the NearConnected state and starts to send INIT_2s. End A has not yet had time to synchronise and receive eight consecutive INIT_1s. It eventually synchronises and receives eight INIT_2s, so it moves to the FarConnected state. In this state End A sends 16 INIT_2s to signal to End B that it is synchronised and then moves on to the Connected state. End B receives eight consecutive INIT_2s, normally the first eight of the sixteen sent, and also moves to the Connected state. The connection has been successfully made in both directions.

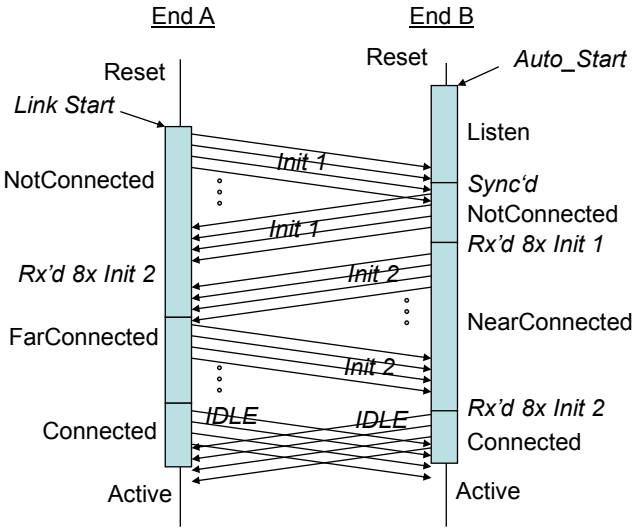

Figure 3-15: Initialisation from Auto-Start through FarConnected State

3.10.2 Initialisation when both ends Link Start

If both ends Link Start at around the same time then the INIT_1, INIT_2 handshake can happen in both directions at the same time. This is illustrated in Figure 3-16. Both ends link start, move to the NotConnected state and begin sending INIT_1s. Both ends synchronise on the received INIT_1s and transition to the NearConnected state once they have received eight consecutive INIT_1s. They then start sending INIT_2s. After eight INIT_2s have been received each end will move to the Connected state, send eight IDLEs and then move to the active state.

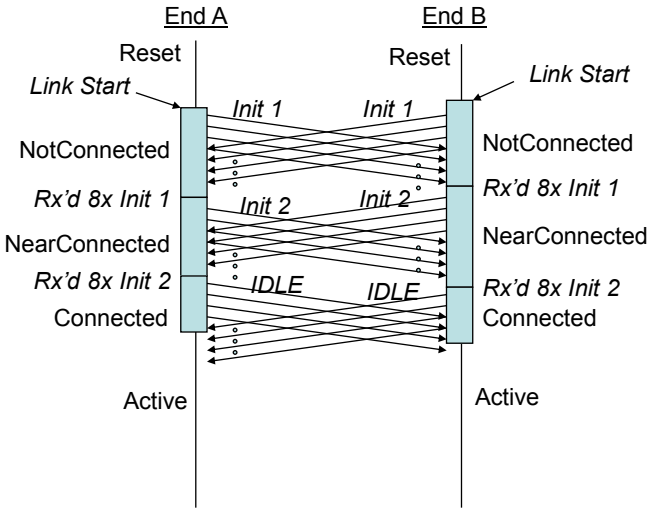

Figure 3-16: Initialisation when both ends Link Start

A similar situation is illustrated in Figure 3-17. This time End A is able to synchronise much quicker than End B. End A will move to the NearConnected state and start sending INIT_2s before End B has received eight consecutive INIT_1s. Eventually End B receives eight consecutive INIT_2s and moves to the FarConnected state. End B then sends 16 INIT_2s and then moves to the Connected state. End A which has been synchronised for a while receives eight consecutive INIT_2s and also moves to the Connected state.

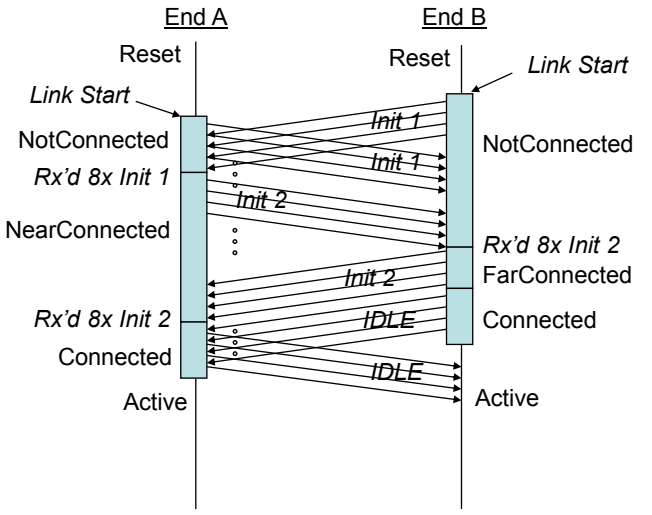

Figure 3-17: Initialisation through FarConnected State

3.10.3 Initialisation with Speed Change

The situation that occurs when a speed change is required is illustrated in Figure 3-18. Both ends go through the initialisation sequence and end up in the Connected state. In this state both ends decide that a speed change is required, move to the Change Speed state, update the operating speed parameter and then do a warm reset. Following the warm reset both ends try to reinitialise running at the new speed.

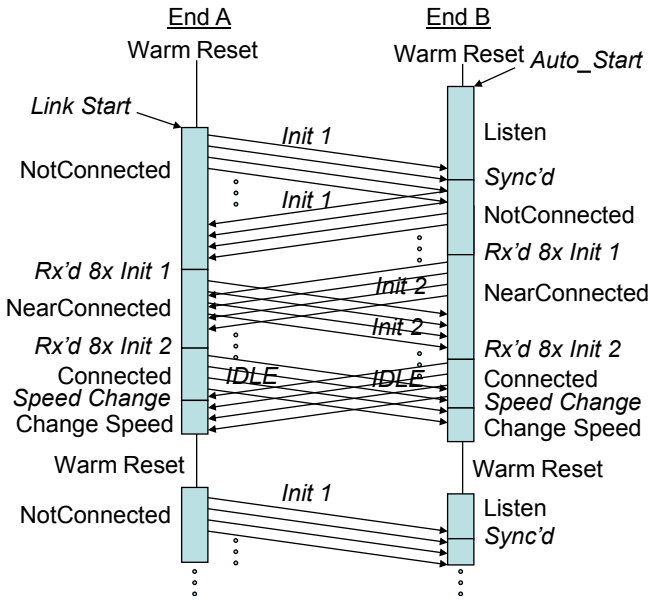

Figure 3-18: Initialisation with Speed Change

3.10.4 Re-Initialisation following Loss-of-Sync

If some form of error occurs and one end of the link loses synchronisation then the link has to be reinitialised. This is illustrated in Figure 3-19. The two state life lines at the left and right hand sides are the Receiver Synchronisation state machine (see Figure 3-11). Both ends are in the Active state sending and receiving information. It is assumed that the Link Start bit is set at both ends of the link. End A then loses synchronisation, possibly going through the CheckSync state. The loss of sync condition is flagged to the Link state machine and it moves from Active to Initialise state. End A then sends INIT_1s as part of the initialisation handshake. End B which is still active receives the INIT_1, indicating that End A is reinitialising. End B thus moves to the initialisation state and starts sending INIT_1s. Eventually end A will resynchronise and the link will be recovered. Note that End B does not

lose synchronisation during this re-initialisation process. Recovery from loss of sync may take just a short time, less than 1 us.

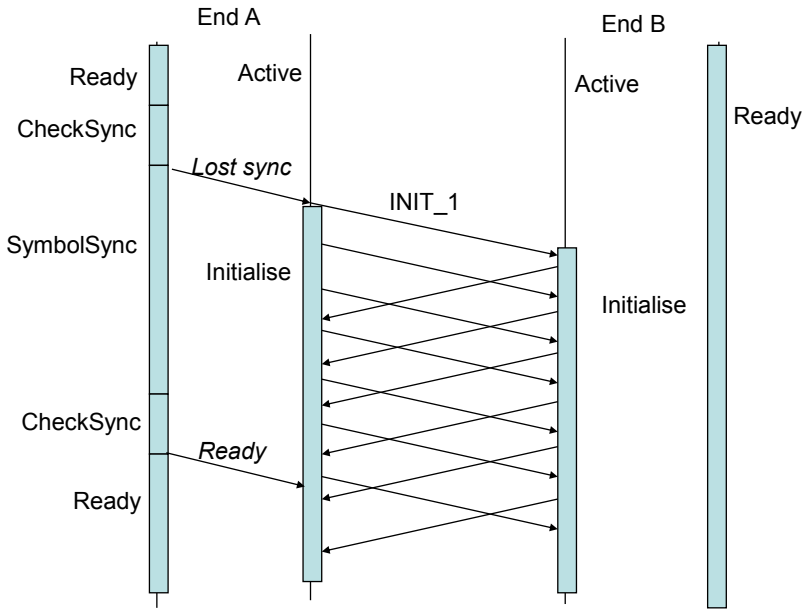

Figure 3-19: Re-Initialisation after Loss of Sync

3.11 POLARITY CHANGE

3.11.1 The Need for Polarity Change

With very high speed signals, such as those used in SpaceFibre, it is essential to ensure good signal integrity. The use of controlled impedance and differential signals goes a long way to achieving this but brings its own difficulties. The two lines of the differential signal have to be tracked across a PCB from one chip to another or from one chip to a connector. The two tracks must follow the same path being kept with a constant separation to maintain the required impedance and must be the same length. This is relatively straightforward if the two signals ca be routed directly to the connector (or other chip) without having to cross over the two signals as would be necessary if a left-hand signal of a differential pair has to go to a right-hand pin and vice versa.

Allowing polarity swapping simplifies this PCB layout problem. The left-hand signal is routed to the left-hand pin regardless of whether this causes a polarity swapping of the differential signal. It is then up to the receiver to sort out any polarity swapping that has occurred. There are four possibilities:

- No polarity swap at transmitter or receiver: in this case there is no need to change the polarity of the receiver input.
- Polarity swap at both transmitter and receiver: in this case there is no need to change the polarity of the receiver input, since one polarity swap cancels out the other.
- No polarity swap at transmitter and polarity swap at receiver: in this case the signal at the receiver input has the wrong polarity so it must be changed.
- Polarity swap at transmitter and no polarity swap at receiver: in this case the signal at the receiver input has the wrong polarity so it must be changed.

The receiver has to detect whether the signal arriving at the receiver is of the wrong polarity. If it is then the polarity of the receiver input has to be changed.

3.11.2 Polarity Detection

Polarity detection can be done using certain features of the 8B/10B 10-bit symbols.

The inverse of the 7-bit comma code (0011111_b) is also a comma code (1100000_b). This means that it is possible to detect a comma regardless of whether the polarity of the receiver input is correct or inverted.

Some data 10-bit codes have a bitwise inverse which is a valid data code, *e.g.* D21.5 is the bitwise inverse of D10.2. If one of these characters is used in the INIT_1 sequence then it will be possible to determine whether the receiver input is inverted or not. Consider the following definition for the INIT_1 Ordered set:

INIT_1 = K28.5, D10.2, Dw.x, D y.z

Where Dw.x and Dy.z are any data character.

K28.5 contains a comma code enabling the INIT_1 ordered set to be synchronised, regardless of receiver input signal polarity. The character immediately following the comma code will be D10.2 if the polarity is correct and D21.5 if the polarity is inverted. All the receiver has to do to detect polarity is to synchronise on comma and then look at the value of the following data character.

D10.2 is chosen as the data character to use to represent INIT_1 because:

- It contains a large number of transitions
- Its bitwise inverse is a valid data character
- The same code generated is generated for D10.2 regardless of current running disparity

The INIT_2 ordered set is constructed with similar characteristics to allow polarity detection and correction if INIT_2s are the first ordered set received after bit and symbol synchronisation.

- INIT_1 = K28.5, D10.2, D0.1, Dm.s
- INIT_2 = K28.5, D10.2, D0.2, Dm.s

Where Dm.s contains the maximum operating speed of the link interface that sent the INIT_1 or INIT_2 ordered set.

3.11.3 Changing Polarity

Having detected the polarity of the receiver input it is simple to change its polarity of necessary. A method for doing this on the serial data is shown in Figure 3-20. An XOR gate is used to provide the required inversion when the Invert signal is asserted.

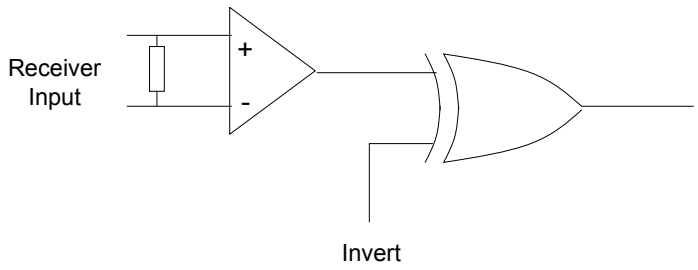

Figure 3-20: Inversion of Receiver Input

In practice it is better to do this after symbol synchronisation before the 8B/10B decoder. This has the advantage that any inversion can be done on a symbol boundary. Note that a mechanism must exist to prevent repeated polarity changes being made since the effect of a polarity change will not be detectable until after symbol decoding which may be many cycles later.

3.12 LINK ERROR RECOVERY

Link error recovery is handled by the various state machines in the link interface.

Transient errors are handled by the Receiver Synchronisation state machine, Figure 3-11. One or two errors are ignored in the hope that they are transitory errors and have not otherwise effected the symbol synchronisation. If a comma realignment occurs then the Receiver Synchronisation state machine will indicate that synchronisation is lost if an invalid symbol occurs within four symbols of the comma detection that caused the comma realignment. If several errors occur within a short period of time, or if an invalid symbol occurs within four symbols of comma realignment then the Receiver

Synchronisation state machine signals loss of sync, indicating that a persistent error has occurred and that re-initialisation is required.

In the event of a persistent error re-initialisation is performed by the Link Initialisation state machine, see Figure 3-12 and Figure 3-13.

3.13 POWER MANAGEMENT

There are four power states:

- Active: PLL running, transmitter and receiver enabled, transmitter sending data, ordered sets or control codes.
- Suspended: PLL running, transmitter disabled, receiver enabled.
- Quiescent: PLL switched off, transmitter disabled, receiver disabled.
- Off: complete interface switched off with the exception of possible V_{aux} for out-of-band signalling.

Power management is outside the scope of the current study.

3.14 LOOP-BACK

Loop back facilities are provided for serial loop-back and possibly parallel loop back. The loop-back facilities are illustrated in Figure 3-21. The serial loop-back is feeds the output of the transmitter serialiser into the receiver de-serialiser. At the same time the input from the receiver driver is fed back to the transmitter driver. This provides both local and remote loop back of the serial data. Local loop-back enables user applications to be tested without the need for a SpaceFibre cable or remote unit. Remote loop-back enables data transfer over cables to be tested independent of the operation of remote end of the link, provided that it can be put in serial loop-back mode.

Parallel loop-back is implemented in the parallel data stream. This may not be implemented if it complicates the receiver clock arrangement.

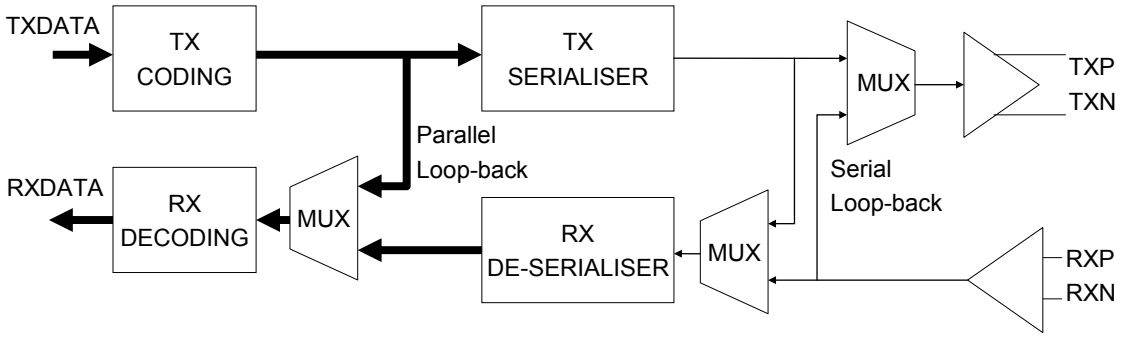

Figure 3-21 Loop-Back Facilities

4. SPACEFIBRE SPECIFICATION

In this section the SpaceFibre specification is provided.

4.1 INTERFACE SPECIFICATION

SpaceFibre defines several interfaces each of which is described in this section.:

- User Interface
- Frame Interface
- SerDes Interface
- Serial Interface

A SpaceFibre CODEC shall implement the Serial Interface.

A SpaceFibre CODEC should implement the SerDes, Frame and User interfaces.

4.1.1 User Interface

The user interface shall be used to pass data and user ordered set for transmission from user logic to the SpaceFibre CODEC and to pass data and user ordered sets received by the SpaceFibre CODEC to user logic.

The user interface shall comprise four parts:

- Transmit Data Frame Input
- Transmit Ordered Set Input
- Receive Data Frame Output
- Receive Ordered Set Output

4.1.1.1 Transmit Data Frame Input

The Transmit Data Frame Input shall contain the signals listed in Table 4-1.

Table 4-1 Transmit Data Frame Interface		
Signal	**Dir**	**Function**
User_Txdata(31:0)	In	User frame data.
User_Txdata_Rdy	In	When asserted a complete frame is ready to transmit. The lowest order byte of the first User_Txdata word holds the frame length and forms part of the start of frame ordered set.
User_Txdata_Read	Out	Read user frame data into the SpaceFibre CODEC for transmission.

4.1.1.2 Transmit Ordered Set Input

The Transmit Ordered Set Input shall contain the signals listed in Table 4-2.

Table 4-2 Transmit Ordered Set Interface		
Signal	**Dir**	**Function**
User_Tx_Ord_Set(31:0)	In	User ordered set data. Bits 31:24 should be set to K28.5 as the ordered set must have a comma code as the most significant byte.
User_Tx_Ord_Set_Rdy	In	When User_Tx_Ord_Set_Rdy is asserted then the User_Tx_Ord_Set data is valid. User ordered sets are transmitter by the SpaceFibre CODEC when the link initialisation state machine has connected
User_Tx_Ord_Set_Read	Out	Asserted when an ordered set has been read into the SpaceFibre CODEC for transmission.

4.1.1.3 Receive Data Frame Output

The Receive Data Frame Input shall contain the signals listed in Table 4-3.

Table 4-3 Receive Data Frame Interface		
Signal	**Dir**	**Function**
User_Rxdata(31:0)	Out	Received frame data.
User_Rxdata_SOF	Out	When asserted the frame data on User_Rxdata is a start of frame
User_Rxdata_EOF	Out	When asserted the frame data on User_Rxdata is an end of frame
User_Rxdata_Valid	Out	When asserted the User_Rxdata, User_Rxdata_SOF and User_Rxdata_EOF outputs are valid.
User_Rx_Out_Of_Frame_Error	Out	Asserted when a character is received when not expected, i.e. an end of frame is received when a start of frame is expected.
User_Frame_Length_Error	Out	Asserted when a frame is terminated with an end of frame ordered set before the complete frame length has been received.

4.1.1.4 Receive Ordered Set Output

The Receive Ordered Set Input shall contain the signals listed in Table 4-4.

Table 4-4 Receive Ordered Set Interface		
Signal	**Dir**	**Function**
User_Rx_Ord_Set(31:0)	Out	Received ordered set data.
User_Rx_Ord_Set_Valid	Out	When asserted the User_Rx_Ord_Set output is valid.

4.1.2 Data Interface

The Data interface shall pass complete data and idle frame and user ordered sets between the Framing and Coding and Link Control parts of the SpaceFibre CODEC.

4.1.2.1 Transmit Data Input

The Transmit Data Input shall contain the signals listed in Table 4-5.

Table 4-5 Transmit Data Interface

Signal	Dir	Function
Frm_Txdata(31:0)	In	Data to send.
Frm_Tx_Ord_Set	In	When asserted Frm_Txdata contains an ordered set for transmission. When de-asserted Frm_Txdata contains frame data for transmission.
Frm_Txdata_Valid	In	Indicates that Frm_Txdata and Frm_Tx_Ord_Set are valid.
Frm_Txdata_Ack	Out	Acknowledge the frame data for transmission.

4.1.2.2 Receive Data Output

The Receive Data Frame Input shall contain the signals listed in Table 4-6.

Table 4-6 Receive Data Interface

Signal	Dir	Function
Frm_Rxdata(31:0)	Out	Received frame data.
Frm_Rx_Ord_Set	Out	When asserted, Frm_Rxdata represents an ordered set
Frm_Rxdata_Valid	Out	When asserted, Frm_Rxdata and Frm_Rx_Ord_Set are valid and the received data is free from link errors.

4.1.3 SerDes Interface

The SerDes Interface shall pass coded, but unsynchronised, symbols between the Coding and Link Control, and Serialisation parts of the SpaceFibre CODEC.

The SerDes interface shall comprise a Transmit SerDes interface and a Receive SerDes interface.

The SerDes interfaces shall be 10-bit, 20-bit or 40-bit wide.

4.1.3.1 Transmit SerDes Output

The Transmit SerDes Output shall contain the signals listed in Table 4-7, Table 4-8, or Table 4-9.

Table 4-7 Transmit SerDes Interface (10-bit)		
Signal	**Dir**	**Function**
SerDes_Txdata(9:0)	In	10-bit wide data containing one symbol for transmission.

Table 4-8 Transmit SerDes Interface (20-bit)		
Signal	**Dir**	**Function**
SerDes_Txdata(19:0)	In	20-bit wide data containing two symbols for transmission.

Table 4-9 Transmit SerDes Interface (40-bit)		
Signal	**Dir**	**Function**
SerDes_Txdata(39:0)	In	40-bit wide data containing four symbols for transmission.

4.1.3.2 Receive SerDes Input

The Receive SerDes Input shall contain the signals listed in Table 4-10, Table 4-11, or Table 4-12.

Table 4-10 Receive SerDes Interface (10-bit)		
Signal	**Dir**	**Function**
SerDes_Rxdata(9:0)	Out	10-bits of received data. This data is NOT symbol synchronised i.e. the 10-bits can contain some bits from one symbol and the rest of the bits from the next symbol.

Table 4-11 Receive SerDes Interface (20-bit)		
Signal	**Dir**	**Function**
SerDes_Rxdata(19:0)	Out	20-bits of received data. This data is NOT symbol synchronised i.e. the 20-bits can contain some bits from one symbol, the following complete symbol, and the rest of the bits from the next symbol.

Table 4-12 Receive SerDes Interface (40-bit)		
Signal	**Dir**	**Function**
SerDes_Rxdata(39:0)	Out	40-bits of received data. This data is NOT symbol synchronised i.e. the 40-bits can contain some bits from one symbol, the following complete three symbols, and the rest of the bits from the next symbol.

4.1.4 Serial Interface

The Serial Interface shall pass serial data out of and into the SpaceFibre CODEC.

The serial interface shall comprise a transmitter serial output and a receiver serial input.

4.1.4.1 Transmit Serial Output

The Transmit Serial Output shall contain the signals listed in Table 4-13.

Table 4-13 Transmit Serial Output	
Signal	**Function**
Txp	Positive side of the differential serial transmitter output.
Txn	Negative side of the differential serial transmitter output.

4.1.4.2 Receive Serial Input

The Receive Serial Input shall contain the signals listed in Table 4-14.

Table 4-14 Receiver Serial Input	
Signal	**Function**
Rxp	Positive side of the differential serial receiver input.
Rxn	Negative side of the differential serial receiver input.

4.2 FUNCTIONAL SPECIFICATION

In this section the functional specification for SpaceFibre is presented.

4.2.1 Data Words and Characters

Information shall be presented to the SpaceFibre CODEC for transmission 32-bits at a time.

Information received by the SpaceFibre CODEC shall be passed to the user logic 32-bits at a time.

A character shall contain either 8-bits of user information or a control code.

A data word shall comprise four characters containing a total of 32-bits of user information.

4.2.2 Ordered Sets

A comma is a control character (K28.5).

An ordered set is a group of four characters starting with a comma (K28.5).

There are several types of ordered set supported or used by SpaceFibre

- Link-level ordered sets
- Power management ordered sets
- Reset ordered sets
- Framing ordered sets
- Flow control ordered sets
- User ordered sets

The link-level ordered sets are used to initialise SpaceFibre link and are defined in Table 4-15 below.

Table 4-15: Link Layer Ordered Sets		
Name	**Ordered Set**	**Function**
SKIP	Comma, SKIP, Count MS, Count LS K28.5, D0.0, cnt_ms, cnt_ls	Send every N ordered sets to support the receiver elastic buffer operation and skip-tick indication. N must be less than or equal to 5000.
IDLE	Comma, IDLE, 0, 0 K28.5, D0.1, D0.0, D0.0	Sent whenever there is no data frame, idle frame or other ordered set to send. It keeps the link active.
INIT_1	Comma, INIT, 1, Speed K28.5, D10.2, D0.1, speed	Send as part of the initialisation handshake. If received at any other time causes a re-initialisation.
INIT_1 Inverted	K28.5, D21.5, D0.6, inv_speed	Sent as part of the initialisation handshake if the signals are inverted.
INIT_2	Comma, INIT, 2, Speed K28.5, D10.2, D0.2, speed	Send as part of the initialisation handshake.

The Power Management and Reset ordered sets have not yet been fully defined but are listed in Table 4-16 to give an indication of their intended function.

Table 4-16: Power Management and Reset Ordered Sets		
Name	**Ordered Set**	**Function**
SUSPEND	Comma, SUSP, SUSP, SUSP	Indicates that transmitter is moving to the Suspend state and will temporarily stop sending data.
STANDBY	Comma, STANDBY, STANDBY, STANDBY	Commands the receiver to enter the Standby state once it has finished sending the current data frame.
POWER OFF	Comma, OFF, OFF, OFF	Commands the receiving system to power down.
WARM RESET	Comma, WARM_RST, WARM_RST, WARM_RST	Commands the receiving SpaceFibre interface to perform a warm reset.
COLD RESET	Comma, COLD_RST, COLD_RST, COLD_RST,	Commands the receiving SpaceFibre interface to perform a cold reset.

Data framing ordered sets provide for management of the data frames and idle frames being set across the link. They are listed in Table 4-17. This table is an initial table only. Support for error recovery, frame retry and redundancy and also for quality of service support have yet to be defined.

Table 4-17: Data Framing Ordered Sets		
Name	**Ordered Set**	**Function**
SDF	Comma, SDF, QoS, Word Count K28.5, D0.2, QoS, Len	Start of Data Frame. Contains type of frame, quality of service information and length of frame.
SIF	Comma, SIF, QoS, Word Count K28.5, D0.3, QoS, Len	Start of Idle Frame. Contains type of frame, quality of service information and length of frame.
EOF	Comma, EOF, CRC MS, CRC LS K28.5, D0.4, crc_ms, crc_ls	End of Frame. Contains CRC for frame.
EEF	Comma, EEF, CRC MS, CRC LS K28.5, D0.5, crc_ms, crc_ls	Error End of Frame. Indicates that the frame was terminated early for some reason.

Flow control ordered sets support flow control across virtual channels. The flow control ordered sets are listed in Table 4-18.

Table 4-18: Flow Control Ordered Set		
Name	**Ordered Set**	**Function**
FCT	Comma, FCT, Sequence No., Channel No. K28.5, D0.6, Seq, Ch	Flow Control Token Indicates that the receive buffer for a specific virtual channel has room for another complete data frame. Sequence number increments for each new FCT sent related to a specific channel. Channel number identifies the virtual channel which this FCT is for.

4.2.3 Data Framing

4.2.3.1 Frames

A frame shall contain user data or idle data.

A frame shall start with a start of data frame (SDF) ordered set or a start of idle frame (SIF) ordered set.

A frame shall end with an end of frame (EOF) ordered set or an error end of frame (EEF) ordered set.

4.2.3.2 Data Frames

A data frame shall contain zero or more data words.

There shall be a maximum of 255 data words of 32-bits each in a data frame.

The word count field in the start of data frame (SDF) shall contain the number of data words in the frame.

The end of frame shall contain a 16-bit CRC covering the data in the data frame.

The data frame is illustrated in Figure 4-1.

31 24	23 16	15 8	7 0
COMMA	SDF/SIF	QoS	Word Count
DATA 1 MS	DATA 1	DATA 1	DATA 1 LS
DATA 2 MS	DATA 2	DATA 2	DATA 2 LS
...	...	...	...
DATA N MS	DATA N	DATA N	DATA N LS
COMMA	EOF/EEF	CRC_MS	CRC_LS

Figure 4-1 Data Frame Format

4.2.3.3 Idle Frames

An idle frame shall be sent whenever there is no data frame to send.

An idle frame shall contain zero or more data words set to zero (subsequently scrambled). The data words set to zero in an idle frame are known as idle words, which are distinct from the idle ordered set.

There shall be a maximum of 255 idle words in an idle frame.

The word count field in the start of idle frame (SIF) shall be set to 255 and is the maximum number of idle words in the idle frame. It does not represent that actual number of words in the idle frame

The end of frame in an idle frame can contain a 16-bit CRC covering the idle words in the idle frame.

An idle frame shall be terminated as soon as there is user data to send. This means that the length of the idle frame can contains zero to 255 idle words.

An idle frame shall contain at least the start of idle frame (SIF) and the end of frame (EOF).

An idle frame is illustrated in Figure 4-2.

31 24	23 16	15 8	7 0
COMMA	SDF/SIF	0	255
IDLE 1 MS	IDLE 1	IDLE 1	IDLE 1 LS
IDLE 2 MS	IDLE 2	IDLE 2	IDLE 2 LS
...	...	...	...
IDLE N MS	IDLE N	IDLE N	IDLE N LS
COMMA	EOF/EEF	CRC_MS	CRC_LS

Figure 4-2 Idle Frame Format

4.2.3.4 Transmit Data Frame Input

When the user logic has data ready to send in a data frame, it shall indicate this to the SpaceFibre CODEC and pass the CODEC the data for the frame.

4.2.3.5 Transmit Idle Frames

When the SpaceFibre CODEC has finished sending a data frame and there is no more data ready to send then it shall send an idle frame immediately after the last data frame.

4.2.3.6 Receive Data Frame Output

Received data frames (from SpaceFibre) shall be passed to the user logic.

4.2.3.7 Receive Idle Frame Removal

Received idle frames (from SpaceFibre) shall be discarded.

4.2.4 Scrambling/ Descrambling

4.2.4.1 Scrambling

The data field of data frames and idle frames shall be scrambled prior to transmission of the frame by bit wise multiplication of the data with a sequence of random numbers produced from a scrambling polynomial.

The scrambling polynomial to use shall be $G(x) = X^{16} + X^5 + X^4 + X^3 + 1$

The seed for the scrambler shall be 0xffff i.e. all flip-flops in the random number generator are set to one.

The scrambler shall be re-seeded at the start of every new data or idle frame.

The single bit output sequence from the random number generator shall be XORed with the bit sequence from the data word.

The least-significant bit of the most-significant character in each word shall be scrambled first.

This scrambler is illustrated in Figure 4-3, which is included for clarity. The scrambler can be implemented in other ways.

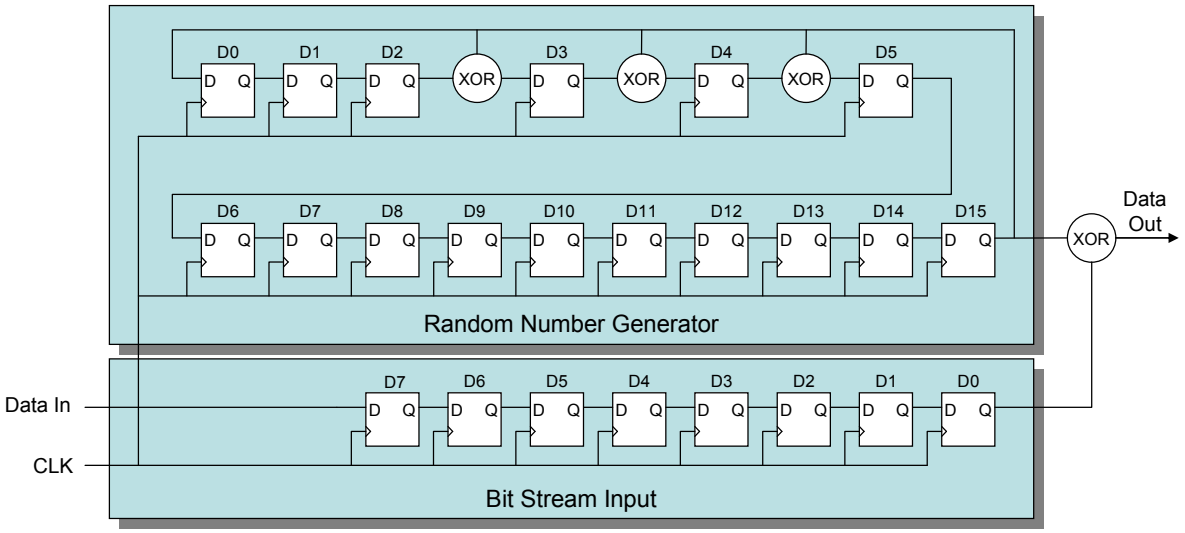

Figure 4-3 Scrambler / De-Scrambler

4.2.4.2 Descrambling

When data frames are received they shall be descrambled by multiplying (XORing) the received data with the same scrambling sequence as was used for scrambling.

4.2.5 Ordered Set Injection

4.2.5.1 Ordered Set Injection

User ordered sets shall be passed to the SpaceFibre CODEC for transmission via the Transmit Ordered Set interface.

User ordered sets shall be transmitted before any word in a data frame or idle frame but after any other ordered set.

Ordered sets shall be interleaved with data words from data frames or idle frames.

The complete priority of ordered sets will be defined in a subsequent revision of this document.

4.2.5.2 Ordered Set Recovery

Received user ordered sets (from SpaceFibre) shall be passed to the user logic via the Receive Ordered Set interface.

4.2.6 Link Initialisation and Power Management

The Link Initialisation state machine shall control the initialisation of the SpaceFibre link: establishing the connection and responding to power management requests.

4.2.6.1 Link Initialisation State Machine

The state diagram for the link initialisation state machine is illustrated in Figure 4-4.

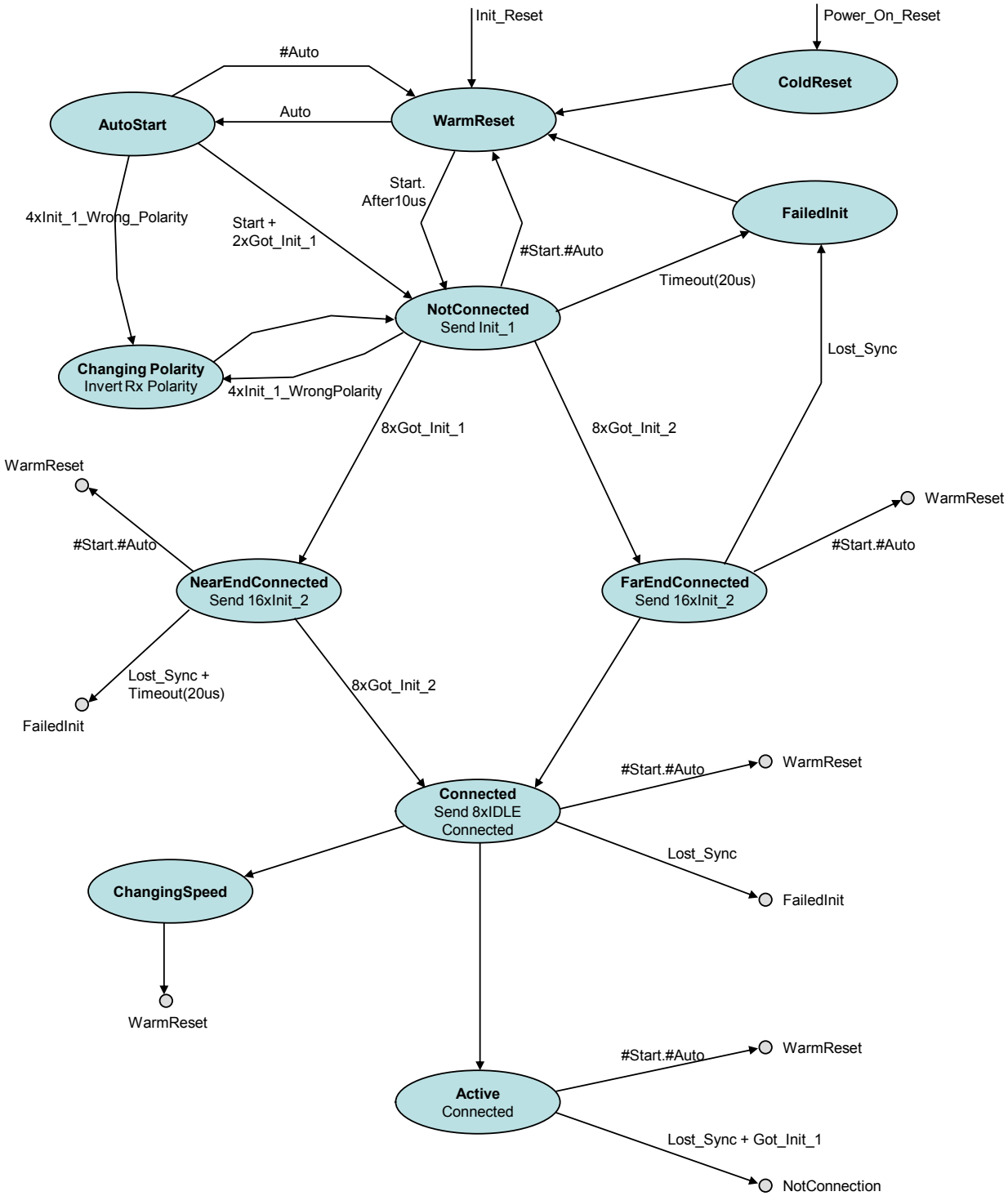

Figure 4-4 Link Initialisation State Machine

4.2.6.1.1 ColdReset State

The ColdReset state shall be entered following power on reset.

When in the ColdReset state the Link Initialisation state machine shall reset the SpaceFibre CODEC and all configuration registers.

After performing the reset of the SpaceFibre CODEC and configuration then Link Initialisation state machine shall move to the WarmReset state.

The ColdReset state is summarised in Table 4-19.

Table 4-19 ColdReset State

State	ColdReset
Entry	From power on reset
Action	Reset SpaceFibre CODEC Reset all configuration registers
Exit	Unconditionally move to the warm reset state

4.2.6.1.2 WarmReset State

The WarmReset state shall be entered on one of the following conditions:

- Following a cold reset
- Following a warm reset
- From the NotConnected, NearEndConnected, FarEndConnected, Connected or Active states when Link Start and Auto-Start signals are both de-asserted
- Unconditionally from the ChangingSpeed state.

When in the WarmReset state the Link Initialisation state machine shall reset the SpaceFibre CODEC and the transmitter, receiver and PLLs shall be disabled.

The Link Initialisation state machine shall remain in the WarmReset state for at least 10 µs +/- 1 µs.

The Link Initialisation state machine shall move to the AutoStart state if the Auto-Start signal is asserted and the Link Start signal is not asserted.

The Link Initialisation state machine shall move to the NotConnected state if the Link-Start signal is asserted.

The WarmReset state is summarised in Table 4-20.

Table 4-20 WarmReset State

State	WarmReset
Entry	Following a cold reset where all registers are set to their power-on state Following a warm reset where configuration registers remain unaltered From the NotConnected, NearEndConnected, FarEndConnected, Connected or Active states when the Link Start and Auto-Start signals are both de-asserted. Unconditionally from the ChangingSpeed state.
Action	Reset the SpaceFibre CODEC Do not reset any configuration registers Rx off, Tx off, PLLs off Remain in this state for minimum of 10 µs (TBC)
Exit	If Link_Start is asserted move to the Initialise state If Auto_Start is asserted move to the Auto_Start state

4.2.6.1.3 AutoStart State

The AutoStart state shall be entered from the WarmReset state when the Auto-Start signal is asserted.

In the AutoStart state the Transmitter shall be disabled and the Receiver, Receiver PLL and Transmitter PLL shall be enabled.

In the AutoStart state the SpaceFibre CODEC shall perform bit, symbol and word synchronisation and shall listen for an INIT_1 ordered set or an inverted INIT_1 ordered set.

If the Link-Start signal is asserted the Link Initialisation state machine shall move from the AutoStart state to the NotConnected state.

If two INIT_1's are received while in the AutoStart state the Link Initialisation state machine shall move to the NotConnected state.

If four inverted INIT_1's are received while in the AutoStart state the Link Initialisation state machine shall move to the ChangingPolarity state.

If the Auto-Start signal is de-asserted the Link Initialisation state machine shall move from the AutoStart state to the NotConnected state.

The AutoStart state is summarised in Table 4-21.

Table 4-21 AutoStart State

State	AutoStart
Entry	From warm reset when Auto-Start signal is asserted
Action	Rx on, Rx PLL on, Tx off, Tx PLL on, Listen for INIT_1 Perform bit and symbol synchronisation
Exit	If Link start is asserted move to the Initialise state If got two INIT_1's is asserted move to the Initialise state If got four inverted INIT_1's is asserted move to the Polarity Change state If Auto_Start signal is de-asserted move to the warm reset state

4.2.6.1.4 NotConnected State

The NotConnected state shall be entered on one of the following conditions:

- From the WarmReset when the Link-Start signal is asserted.
- From the AutoStart state when the Link-Start signal is asserted or when an INIT_1 ordered set has been received.
- From the Active state when synchronisation is lost (Lost-Sync signal asserted) or when an INIT_1 is received indicating loss of synchronisation at the far end of the link.

An initialisation timeout timer shall be used to prevent the SpaceFibre CODEC from remaining in the NotConnected state or NearEndConnected state indefinitely if it fails to connect.

The period of the initialisation timeout interval shall be TBC (much greater than the WarmReset interval).

When the NotConnected State is entered the initialisation timeout timer shall be started.

When in the NotConnected State the transmitter, transmitter PLL, receiver, and receiver PLL shall all be enabled and the transmitter shall send INIT_1 ordered sets continuously.

If at least eight consecutive INIT_1 ordered sets are received the Link Initialisation state machine shall move to the NearEndConnected state.

If at least eight consecutive INIT_2 ordered sets are received the Link Initialisation state machine shall to the FarEndConnected state.

If a timeout occurs the Link Initialisation state machine shall move to the FailedInit state.

If the Link-Start and Auto-Start signals are both de-asserted then the Link Initialisation state machine shall move from the NotConnected state to the WarmStart state.

The NotConnected state is summarised in Table 4-22.

Table 4-22 NotConnected State	
State	NotConnected i.e. Near End Not Connected: Far End Not Connected The near end has not successfully received INIT_1s or INIT_2s from the other end.
Entry	From WarmReset state when the Link Start bit is asserted From AutoStart state when the Link Start bit is asserted From AutoStart state when an INIT_1 has been received From Active state when Lost Sync or an INIT_1 is received indicating loss of sync at far end of link
Action	Start timeout timer (timeout interval much greater than the warm reset time) Send INIT_1 ordered sets continuously
Exit	When 8 consecutive INIT_1 ordered sets have been received move to the NearEndConnected state. when 8 consecutive INIT_2 ordered sets have been received move to the FarEndConnected state. If a timeout occurs move to the FailedInit state.

4.2.6.1.5 NearEndConnected State

The NearEndConnected state shall be entered from the NotConnected state when eight consecutive INIT_1 ordered sets have been received.

When the NearEndConnected State is entered the initialisation timeout timer shall be re-started.

When in the NearEndConnected State the transmitter, transmitter PLL, receiver, and receiver PLL shall all be enabled and the transmitter shall send INIT_2 ordered sets continuously.

At least 16 INIT_2 ordered sets shall be sent before leaving the NearEndConnected state.

If at least eight consecutive INIT_2 ordered sets are received the Link Initialisation state machine shall to the Connected state.

If synchronisation is lost while in the NearEndConnected state (Lost_Sync asserted) or a timeout occurs the Link Initialisation state machine shall move to the FailedInit state.

If the Link-Start and Auto-Start signals are both de-asserted then the Link Initialisation state machine shall move from the NearEndConnected state to the WarmStart state.

The NearEndConnected State is summarised in Table 4-23.

Table 4-23 NearEndConnected State	
State	NearEndConnected i.e. Near End Connected: Far End Not Connected The near end has synchronised and received 8 consecutive INIT_1s. It is not known whether the far end is yet synchronised.
Entry	From NotConnected state when 8 consecutive INIT_1s are received
Action	Start timeout timer (timeout interval much greater than the warm reset time) Send INIT_2s continuously At least 16 INIT_2s must be sent
Exit	When 8 consecutive INIT_2s have been received move to the Connected state If a timeout occurs move to the FailedInit state If a loss_of_sync occurs move to the FailedInit state

4.2.6.1.6 FarEndConnected State

The FarEndConnected state shall be entered from the NotConnected state when eight consecutive INIT_2 ordered sets have been received.

When in the FarEndConnected State the transmitter, transmitter PLL, receiver, and receiver PLL shall all be enabled and the transmitter shall send sixteen INIT_2 ordered sets one after the other.

When in the FarEndConnected State the reception of data frames, idle frames and user ordered sets shall be enabled.

After sending sixteen INIT_2 ordered sets the Link Initialisation state machine shall to the Connected state.

If synchronisation is lost while in the FarEndConnected state (Lost_Sync asserted) the Link Initialisation state machine shall move to the FailedInit state.

If the Link-Start and Auto-Start signals are both de-asserted then the Link Initialisation state machine shall move from the FarEndConnected state to the WarmStart state.

The FarEndConnected State is summarised in Table 4-24.

Table 4-24 FarEndConnected State	
State	FarEndConnected i.e. Near End Not Connected: Far End Connected The far end has synchronised and is now sending INIT_2s The near end did not synchronise on INIT_1s but on INIT_2 so need to make sure that enough INIT_2s are sent so that the far end can get 8 consecutive INIT_2s.
Entry	From NotConnected state when 8 consecutive INIT_2s are received
Action	Send 16 INIT_2s The reception of data frames and user ordered sets is enabled.
Exit	Unconditional transition to Connected state once 16 INIT_2s have been sent If a loss_of_sync occurs move to FailedInit state.

4.2.6.1.7 Connected State

The Connected state shall be entered from the NearEndConnected state when eight consecutive INIT_2 ordered sets have been received.

The Connected state shall be entered from the FarEndConnected state when sixteen INIT_2 oredered sets have been sent.

In the Connected state the transmitter, transmitter PLL, receiver, and receiver PLL shall all be enabled and the transmitter shall send the eight IDLE ordered sets one after the other.

When in the Connected State the reception of data frames, idle and user ordered sets shall be enabled.

The Link Initialisation state machine shall move to the Active state after the eight IDLE ordered sets have been sent.

If synchronisation is lost while in the Connected state (Lost_Sync asserted) the Link Initialisation state machine shall move to the FailedInit state.

If the Link-Start and Auto-Start signals are both de-asserted then the Link Initialisation state machine shall move from the Connected state to the WarmStart state.

The Connected state is summarised in Table 4-25.

Table 4-25 Connected State	
State	Connected i.e. Near End Connected: Far End Connected Both ends are synchronised
Entry	From NearEndConnected state when 8 consecutive INIT_2s have been received From FarEndConnected state when 16 INIT_2s have been sent
Action	Send 8 IDLE ordered sets
Exit	Unconditional transition to Active state after 8 IDLEs have been sent If a loss_of_sync occurs move to the FailedInit state.

4.2.6.1.8 Active State

The Active state shall be entered from the Connected state once eight IDLE ordered sets have been sent.

In the Active state the transmitter, transmitter PLL, receiver, and receiver PLL shall all be enabled

In the Active State the transmission and reception of data frames, idle and user ordered sets shall be enabled.

If synchronisation is lost while in the Active state (Lost_Sync asserted) the Link Initialisation state machine shall move to the NotConnected state.

If an INIT_1 ordered set is received while in the Active state (Lost_Sync asserted) the Link Initialisation state machine shall move to the NotConnected state.

If the Link-Start and Auto-Start signals are both de-asserted then the Link Initialisation state machine shall move from the Active state to the WarmStart state.

The Active state is summarised in Table 4-26.

Table 4-26 Active State	
State	Active
Entry	From Connected state once 8 IDLE ordered sets have been sent.
Action	Rx on, Rx PLL on, Tx on, Tx PLL on, Send and receive data frames, idle frames, and ordered sets
Exit	If Link Start is de-asserted and Auto Start is de-asserted then move to the WarmReset state If there is a Loss of sync then move to the NotConnected state If an INIT_1 Is received then move to the NotConnected state

4.2.6.1.9 ChangingPolarity State

The ChangingPolarity state shall be entered from the NotConnected state when the inverse of the INIT_1 identifier has been received at least four times.

The ChangingPolarity state shall be entered from the AutoStart state when the inverse of the INIT_1 identifier has been received at least four times.

In the ChangingPolarity state the coded data from the receiver shall be inverted, provided that it has not already been inverted since leaving the WarmReset state.

The Link Initialisation state machine shall move from the ChangingPolarity state to the NotConnected state as soon as the polarity of the received coded data has been changed (inverted).

The ChangingPolarity state is summarised in Table 4-27.

Table 4-27 ChangingPolarity State	
State	ChangingPolarity
Entry	From NotConnected state when the inverse of the Init_1 identifier has been received four times. From AutoStart state when the inverse of the Init_1 identifier has been received four times.
Action	If a polarity change has not already taken place since leaving the WarmReset state change the polarity on the receiver. Allows for easy routing of PCB i.e. no need to worry about which pin is + or -.
Exit	Unconditional transfer to the NotConnected state.

4.2.6.1.10 ChangingSpeed State

The ChangingSpeed state shall be entered from the Connected state when far end of the link has signalled in the INIT_1s and/or INIT_2s that it sent, that it can support a higher data rate than the current data rate.

In the ChangingSpeed state the link speed configuration parameter is changed to the highest data rate that can be supported by both ends of the link, as indicated in the INIT_1s and/or INIT_2s it has received.

The Link Initialisation state machine shall move unconditionally from the ChangingSpeed state to the WarmReset state as soon as the link speed parameter has been changed.

The ChangingSpeed state is summarised in Table 4-28.

Table 4-28 ChangingSpeed State	
State	ChangingSpeed
Entry	From Connected state after initialisation handshake when the far end has signalled in INIT_1 and INIT_2 that it can support a higher data rate which is also supported by near end.
Action	Change link speed configuration parameter to the highest data rate that can be supported by both ends
Exit	Unconditional transition to Warm Reset state This results in re-initialisation of the link at the higher data rate

4.2.6.1.11 FailedInit State

The FailedInit state shall be entered from the NotConnected state when a timeout occurs while waiting for eight consecutive INIT_1s, or when synchronisation is lost (Lost_Sync asserted).

The FailedInit state shall be entered from the NearEndConnected state when a timeout occurs while waiting for eight consecutive INIT_1s, or when synchronisation is lost (Lost_Sync asserted).

The FailedInit state shall be entered from the FarEndConnected state when synchronisation is lost (Lost_Sync asserted).

The FailedInit state shall be entered from the Connected state when synchronisation is lost (Lost_Sync asserted).

In the FailedInit state the fact that the link failed to initialise shall be recorded in a status register.

The number of times the link failed to initialise can be recorded.

Whether the Link Initialisation state machine was waiting for INIT_1s or INIT_2s can be recorded.

The Link Initialisation state machine shall move unconditionally from the FailedInit state to the WarmReset state as soon link initialisation failure has been recorded.

The FailedInit state is summarised in Table 4-28.

Table 4-29 FailedInit State	
State	FailedInit
Entry	From NotConnected state when a timeout occurs while waiting for 8 consecutive INIT_1s From NearEndConnected state when a timeout occurs while waiting for 8 consecutive INIT_2s From NotConnected state when synchronisation is lost. From NearEndConnected state when synchronisation is lost. From FarEndConnected state when synchronisation is lost. From Connected state when synchronisation is lost.
Action	Record the fact that the link failed to initialise and whether waiting for INIT_1s or INIT_2s
Exit	Unconditional transition to Warm Reset state The link may then attempt to restart once again depending on configuration settings i.e. if Retry Link Start bit is set (1)

4.2.7 Data Rate Adjustment

The **symbol rate** is the rate at which symbols can be handled in the transmitter and receiver.

The symbol rate of a transmitter at one end of a link and a receiver at the other end of a link might be different due to differences in the local clocks being used at each end of the link. The maximum permitted difference in the symbol rates, when operating at nominally the same data signalling rate (e.g. 2 Gbits/s) shall be 100 parts per million.

Note at a bit rate of 2 GHz this corresponds to a maximum clock difference of 200 kHz.

A receive elastic buffer shall be used to compensate for differences in the clock speeds at each end of the link.

Skip ordered sets shall be inserted regularly in the transmit data stream.

When a skip ordered set is read from a receive elastic buffer that is more than half full the skip character shall be removed from the receive elastic buffer and the following character shall be read from the buffer.

When a skip ordered set is read from a receive elastic buffer that is less than half full the skip character shall be left in the buffer, so that it can be read again the next time the buffer is read. A skip character shall be read at most twice.

A skip ordered set shall be sent every 20,000 symbols.

4.2.8 8B/10B Encoding/Decoding

The SpaceFibre CODEC shall use 8B/10B encoding to encode each 8-bit data character or control code into a 10-bit symbol that is transmitted.

The **current running disparity** is the number of 1's sent compared to the number of 0's.

If the number of 1's sent is one more than the number of 0's then the current running disparity shall be positive.

If the number of 1's sent is one less than the number of 1's then the current running disparity shall be negative.

If the number of 1's and 0's sent is the same, then the current running disparity shall be neutral.

To ensure DC balancing of the transmitted signal account shall be kept of the current running disparity in the transmitter.

If the current running disparity is positive when encoding an 8-bit character or control code then the symbol for that data character or control code which has negative disparity (fewer 1's than 0's) shall be used.

If the current running disparity is negative when encoding an 8-bit character or control code then the symbol for that data character or control code which has positive disparity (more 1's than 0's) shall be used.

To detect disparity errors account shall be kept of the current running disparity.

If the number of 1's received is greater than one more than the number of 0's sent then this shall indicate a disparity error.

If the number of 1's received is lower than one less than the number of 0's sent then this shall indicate a disparity error.

When a symbol is received it shall be decoded into an 8-bit data character or control code using the 8B/10B symbol table.

An **unrecognised symbol** is a symbol that does not appear in the 8B/10B symbol table.

If an unrecognised symbol is received then a symbol error shall be indicated.

The complete 8B/10B coding table shall be included in a later revision of this document.

4.2.9 Serialiser/Deserialiser

10-bit symbols shall be transmitted serially over an appropriate medium.

At the transmitter a serialiser shall be used to convert the parallel 10-bit symbol into a serial bit stream with each bit of the symbol being send one after the other.

The least significant bit of the 10-bit symbol shall be transmitted first. XXX CHECK THIS

At the receiver the incoming bit stream shall be converted to a 10-bit word by sampling the bit stream with a receive clock in a deserialiser.

Note: the deserialiser does not necessarily produce a stream of 10-bit symbols because the boundary of the symbols is not known b y the deserialiser.

4.2.10 Synchronisation

4.2.10.1 Bit Synchronisation

The receive clock used to sample the incoming bit stream, shall be generated by a clock recovery circuit that matches the phase of a local receive clock to the transitions of the incoming bit stream.

Sampling of the bit stream shall be close (+/- 20% TBC) to the centre of the bit period.

The clock recovery shall indicated to the Receive Synchronisation state machine when bit synchronisation is achieved by asserting the LOCKED signal.

4.2.10.2 Symbol Synchronisation

The boundary between symbols shall be determined by detecting the unique comma sequences.

The **Positive Comma** sequence is 0011111

The **Negative Comma** sequence is 1100000

Both positive and negative commas should be detected (TBC) and used for synchronisation.

When a comma sequence is detected the first bit of the following symbol shall be at a position four bits after the end of the comma sequence.

The 10-bit word stream shall be aligned or realigned so that each 10-bit word contains one complete symbol.

The 10-bit word shall be realigned every time a comma sequence is detected in a different position to the position of the last comma detected.

If a realignment is performed, this shall be indicated to the Receive Synchronisation state machine

4.2.10.3 Word Synchronisation

Synchronisation of four consecutive symbols into a group of four symbols, that represent a 32-bit data word or ordered set, shall be performed by so that the symbol containing a comma sequence is the fist (most significant symbol) of the group of four.

4.2.10.4 Receive Synchronisation State Machine

A receive synchronisation state machine shall be used to determine when incoming symbols are properly synchronised.

The state diagram for the receive synchronisation state machine is illustrated in Figure 4-5.

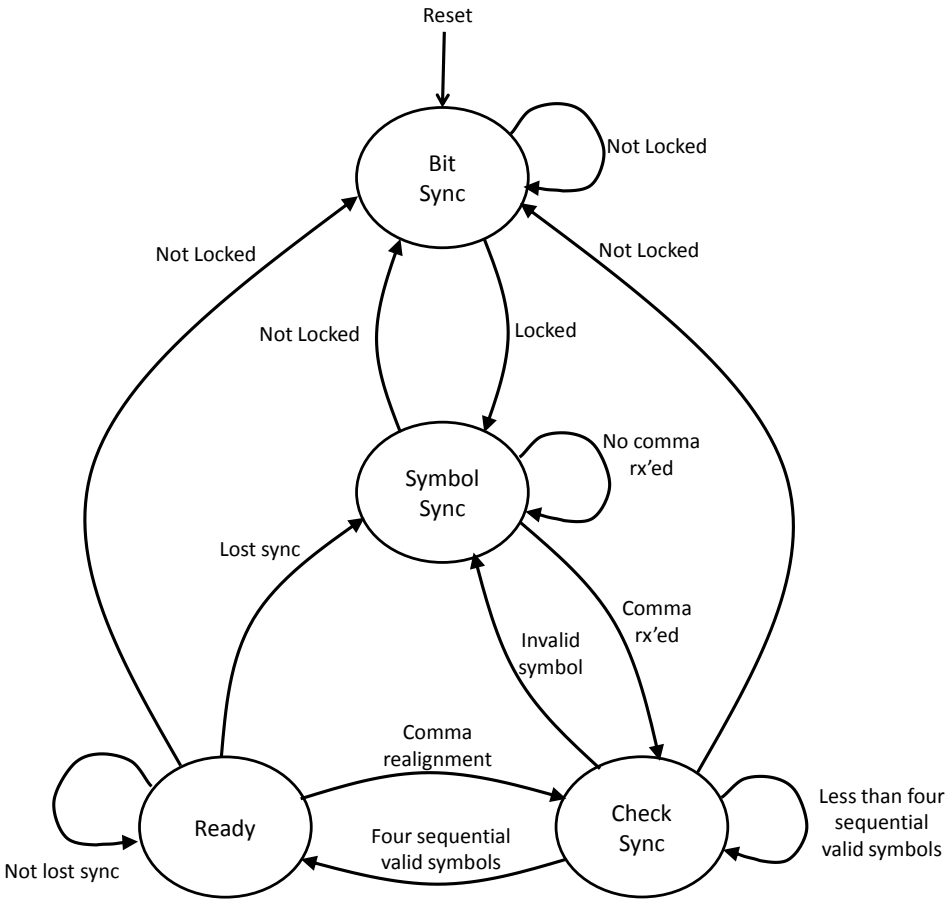

Figure 4-5 Receive Synchronisation State Machine

4.2.10.4.1 Loss of Sync Counter

An **invalid symbol** is either a symbol that contains a disparity error i.e. it results in a running disparity greater than one or less than minus one, or is a symbol that does not occur in the 8B/10B decoding table i.e. is not a valid symbol for an 8-bit data character neither is it a valid symbol for a control code.

A **valid symbol** is one that does not contain a disparity error and is found in the 8B/10B decoding table.

A loss of sync counter shall be used to determine when synchronisation has been lost.

The loss of sync counter shall be incremented by four (TBC) every time an invalid symbol is received.

The loss of sync counter shall be decremented by one every time a valid symbol is received.

The loss of sync counter shall not be decremented below zero.

A loss of synchronisation shall occur when the loss of sync counter reaches a value of 12 (TBC).

The loss of synchronisation shall be flagged to the receive synchronisation state machine.

The loss of sync counter shall be reset to zero when the Receive Synchronisation state machine is in the BitSync, SymbolSync, or CheckSync states.

The loss of sync counter shall be enabled when the Receive Synchronisation state machine is in the Ready state.

4.2.10.4.2 Bit Sync State

The BitSync state shall be entered after power on reset or cold reset of the SpaceFibre CODEC.

The BitSync state shall be entered from the SymbolSync state when bit synchronisation is lost (i.e. LOCKED de-asserted).

The BitSync state shall be entered from the CheckSync state when bit synchronisation is lost (i.e. LOCKED de-asserted).

The BitSync state shall be entered from the Ready state when bit synchronisation is lost (i.e. LOCKED de-asserted).

In the BitSync state the Receiver Synchronisation state machine shall wait for bit synchronisation to be achieved by the Receive Clock Recovery circuit.

In the BitSync state the Receiver Synchronisation state machine shall reset the Loss of Sync Counter to zero.

In the BitSync state the Receiver Synchronisation state machine shall indicate Lost Sync to the Link Initialisation state machine.

When bit synchronisation is achieved the Receiver Synchronisation state machine shall move to the SymbolSync state.

The BitSync state is summarised in Table 4-30.

Table 4-30 BitSync State

State	BitSync
Entry	From power on reset From the SymbolSync state when loss of bit synchronisation (Not Locked) is detected From the CheckSync state when loss of bit synchronisation (Not Locked) is detected From the SymbolSync state when loss of bit synchronisation (Not Locked) is detected
Action	Wait for bit synchronisation. Reset loss of sync counter to zero. Indicate Lost Sync to the Link Initialisation state machine
Exit	When bit synchronisation is achieved move to the SymbolSync state

4.2.10.4.3 SymbolSync State

The SymbolSync state shall be entered from the BitSync state when the bit synchronisation is achieved e.g. when the clock recovery PLL has locked onto the incoming bit stream.

The SymbolSync state shall be entered from the CheckSync state when an invalid symbol is detected.

The SymbolSync state shall be entered from the Ready state when a loss of synchronisation occurs as detected by the Loss of Sync Counter.

In the SymbolSync state the receiver shall listen for commas.

In the SymbolSync state the Receiver Synchronisation state machine shall reset the Loss of Sync Counter to zero.

In the SymbolSync state the Receiver Synchronisation state machine shall indicate Lost Sync to the Link Initialisation state machine.

When a comma is received the Receiver Synchronisation state machine shall move from the SymbolSync state to the CheckSync state.

The SymbolSync state is summarised in Table 4-31.

Table 4-31 SymbolSync State	
State	SymbolSync
Entry	From power on reset From the CheckSync state when an invalid symbol is detected From the Ready state when synchronisation is lost
Action	Listen for commas Reset loss of sync counter to zero. Indicate Lost Sync to the Link Initialisation state machine
Exit	When a comma is received move to the CheckSync state

4.2.10.4.4 CheckSync State

The CheckSync state shall be entered from the SymbolSync state when a comma is received.

The CheckSync state shall be entered from the Ready state when comma realignment occurs.

In the CheckSync state the receiver shall receive symbols.

In the CheckSync state the Receiver Synchronisation state machine shall reset the Loss of Sync Counter to zero.

When an invalid symbol is received the Receiver Synchronisation state machine shall move from the CheckSync state to the SymbolSync state.

When four consecutive valid characters have been received the Receive Synchronisation state machine shall move from the CheckSync state to the Ready state.

The CheckSync state is summarised in Table 4-32.

Table 4-32 CheckSync State	
State	CheckSync
Entry	From the SymbolSync state when a comma is detected From the Ready state when comma realignment occurs
Action	Receive symbols. Reset loss of sync counter to zero.
Exit	When an invalid symbol is received move to the SymbolSync state. When four consecutive valid characters have been received move to the Ready state.

4.2.10.4.5 Ready State

The Ready state shall be entered from the CheckSync state when four consecutive valid symbols have been detected.

In the CheckSync state the receiver shall receive symbols.

In the CheckSync state the Receiver Synchronisation state machine shall enable the Loss of Sync Counter.

When comma realignment occurs the Receiver Synchronisation state machine shall move from the Ready state to the CheckSync state.

When the Loss of Sync Counter indicates that synchronisation has been lost the Receive Synchronisation state machine shall move from the Ready state to the SymbolSync state.

The Ready state is summarised in Table 4-33.

Table 4-33 Ready State	
State	Ready
Entry	From the CheckSync state when four consecutive valid symbols have been detected.
Action	Enable loss of sync counter. Indicate that the link is synchronised to the Link Initialisation state machine.
Exit	When comma realignment occurs move to the CheckSync state. When the loss of sync counter indicates that synchronisation has been lost, move to the SymbolSync state.

4.2.11 Inversion

The received symbols shall be bit-wise inverted if requested by the link initialisation state machine.

4.2.12 Electrical SpaceFibre Medium

4.2.12.1 Electrical SpaceFibre Driver and Receiver

The driver and receiver for SpaceFibre operation over copper shall use Current Mode Logic (CML).

4.2.12.2 Electrical SpaceFibre PCB Tracks

The PCB tracks for electrical SpaceFibre shall be 100 ohms differential impedance.

Two pairs of differential PCB tracks shall be used for bi-direction SpaceFibre; one pair for each direction.

4.2.12.3 Single-ended Electrical Connectors

The single ended electrical SpaceFibre connectors shall be 50 ohm SMA connector.

4.2.12.4 SpaceFibre Electrical Cables

The electrical SpaceFibre cable shall be 50 ohm coaxial cable.

Two coax cables shall be used for for bi-direction SpaceFibre; one cable for each direction.

4.2.12.5 Differential Electrical Connectors

TBA

4.2.12.6 Differential Electrical Cables

TBA

4.2.13 Fibre Optic Driver and Receiver

4.2.13.1 Fibre Optic Driver and Receiver

TBA

4.2.13.2 Fibre Optic Connectors

TBA

4.2.13.3 Fibre Optic Cables

TBA

4.2.14 Parallel Loopback

A parallel loopback facility should be provided in the SpaceFibre CODEC for test purposes.

When enabled, the parallel loopback shall connect the 10-bit symbols from the transmitter directly into the path of the receiver at point after 10-bit symbols have been synchronised.

4.2.15 Serial Loopback

A serial loopback facility should be provided in the SpaceFibre CODEC for test purposes.

The serial loopback shall connect the bit stream output from the serialiser in the transmitter directly into the serial input of the deserialiser in the receiver.

5. SPACEFIBRE FLOW CONTROL AND VIRTUAL CHANNELS

This section describes the flow control mechanism used in SpaceFibre and how this interacts with SpaceFibre virtual channels.

Flow control and virtual channels are not part of the SpaceFibre CODEC but are essential elements of a SpaceFibre system.

5.1 VIRTUAL CHANNELS

A **virtual channel** is a unidirectional data connection between an identified pair of buffers, one at either end of a SpaceFibre link, through which data flows from the source buffer to the destination buffer.

A **source virtual channel buffer** is a buffer associated with a virtual channel that is providing data to be sent across a SpaceFire link.

A **destination virtual channel buffer** is a buffer associated with a virtual channel that is receiving data from a SpaceFire link.

A maximum of 256 virtual channels shall be supported.

A specific virtual channel shall be identified by a virtual channel number in the range 0 to 255.

A source virtual channel buffer shall sent data to a destination virtual channel buffer with the same virtual channel number as the source virtual channel.

5.2 FLOW CONTROL

Flow control shall be performed across a SpaceFibre Link.

The SpaceFibre flow control mechanism shall control the flow of data from a source virtual channel buffer at one end of a SpaceFibre link to a destination virtual channel buffer at the other end of the link with the same virtual channel number.

The destination virtual channel buffer shall send flow control ordered sets to the source virtual channel buffer to indicate that it has room for more frames of data.

One flow control ordered set shall be sent every time room is made in the destination virtual channel buffer for another complete frame of data.

The space in the destination virtual channel buffer related to a flow control ordered set shall be reserved for a complete frame of data once the flow control token has been sent.

The source virtual channel buffer shall only send a frame of data when there is room in the destination virtual channel buffer.

The flow control ordered set (illustrated in Table 4-18) shall contain the virtual channel number for the channel it is providing flow control.

The flow control ordered set shall contain a sequence number.

The flow control ordered set sequence number shall be reset to zero when the SpaceFibre CODEC is reset.

The flow control ordered set sequence number shall be incremented every time a flow control ordered set is sent for a particular virtual channel number.

After reset one or more flow control tokens shall be sent immediately reserving all the space in the destination virtual channel buffer.

The flow control mechanism in the source virtual channel shall keep track of the amount of space available in the destination virtual channel buffer so that it is able to send a data frame as soon as it has one to send.

The flow control mechanism in the source virtual channel shall keep a count of the number of data frames it is allowed to send in a source virtual channel flow control counter.

The flow control mechanism in the source virtual channel shall keep track of the sequence number of the last flow control ordered set received for that virtual channel in a source virtual channel sequence number register.

The source virtual channel flow control counter shall be reset to zero when the SpaceFibre CODEC is reset.

The source virtual channel sequence number register shall be reset to zero when the SpaceFibre CODEC is reset.

The source virtual channel flow control counter shall be incremented by one when a flow control ordered set is received for that virtual channel with a sequence number which is one more than the value in the corresponding source virtual channel sequence number register.

If the source virtual channel receives a flow control ordered set with the same sequence number as that contained in the source virtual channel sequence number register, then the flow control ordered set shall be ignored.

When a flow control ordered set has been lost and a subsequent ordered set arrives for that virtual channel, the source virtual channel flow control counter shall be incremented by the difference

between the sequence number in the received flow control ordered set and the value in the corresponding source virtual channel sequence number register.

A timer, the destination watchdog timer, associated with each destination virtual channel buffer shall be used to recover from lost flow control ordered sets.

The destination watchdog timer shall be reset every time a data frame is received.

If the destination watchdog timer expires, indicating that no data frame has been received within a certain time period since the last flow control ordered set was sent, then a flow control ordered set shall be sent with the same sequence number as the previously send flow control ordered set for that virtual channel.