

Technology

Techniques for implementing gigabit-serial I/O

Real-World Serial I/O

In the previous chapters, we examined some of the challenges faced by input/output (I/O) designers. And we looked at some of the advantages serial I/O has to offer. But how does a design engineer actually make use of serial I/O technology? Before design can begin, we need to know what's available to help implement serial I/O solutions. We need to examine some serial building block devices to see if the tools are there for real-world implementation.

Gigabit-Serial Implementations

In this chapter we will look at some of the technologies involved with multi-gigabit links. We'll look at the Serializer/Deserializer (SERDES), its basic building blocks, and learn how all the speed is achieved (Figure 3-1). We will also review the format of the serial stream from both logical and physical points of view. This information will give us a foundation for planning gigabit-serial I/O designs.

SERDES

History of SERDES and CDR

Serial-to-parallel and parallel-to-serial conversions have been a part of I/O design from the beginning. So has the idea of recovering a clock, or “locking a clock to an incoming stream.” So why has the SERDES suddenly become so important?

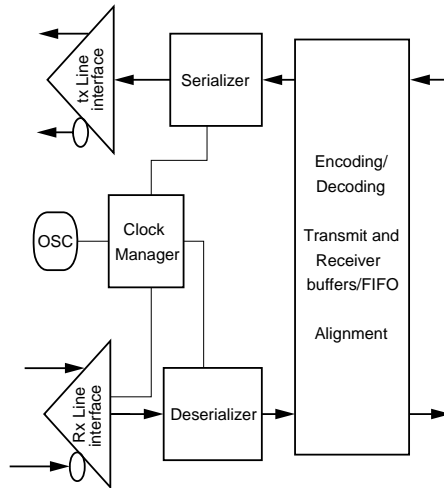


FIGURE 3-1: SERDES Block Diagram

As integrated circuit (IC) geometry grew smaller and maximum toggle rate (F_{max}) increased, the need for I/O bandwidth exploded. In fact, some developments allowed for I/O frequency even faster than F_{max} .

F_{max} : Maximum toggle rate of a flip-flop in a given technology or part.

Basic Theory of Operations and Generic Block Diagram

Let's look at the basic building blocks of a SERDES (Figure 3-2).

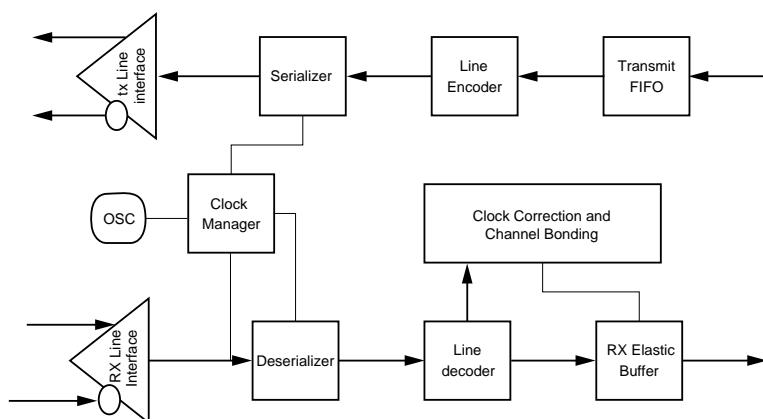


FIGURE 3-2: SERDES Generic Block Diagram

- **Serializer:** Takes n bits of parallel data changing at rate y and transforms them into a serial stream at a rate of n times y .
- **Deserializer:** Takes serial stream at a rate of n times y and changes it into parallel data of width n changing at rate y .
- **Rx (Receive) Align:** Aligns the incoming data into the proper word boundaries. Several different mechanisms can be used from automatic detection and alignment of a special reserved bit sequence (often called a comma) to user-controlled bit slips.
- **Clock Manager:** Manages various clocking needs including clock multiplication, clock division, and clock recovery.
- **Transmit FIFO (First In First Out):** Allows for storing of incoming data before transmission.
- **Receive FIFO:** Allows for storing of received data before removal; is essential in a system where clock correction is required.
- **Receive Line Interface:** Analog receive circuitry includes differential receiver and may include active or passive equalization.
- **Transmit Line Interface:** Analog transmission circuit often allows varying drive strengths. It may also allow for pre-emphasis of transitions.
- **Line Encoder:** Encodes the data into a more line-friendly format. This usually involves eliminating long sequences of non-changing bits. May also adjust data for an even balance of ones and zeros. (This is an optional block sometimes not included in a SERDES.)
- **Line Decoder:** Decodes from line encoded data to plain data. (This is an optional block that is sometimes done outside of the SERDES.)
- **Clock Correction and Channel Bonding:** Allows for correction of the difference between the transmit clock and the receive clock. Also allows for skew correction between multiple channels. (Channel bonding is optional and not always included in SERDES.)

Other possible functions can be included such as cyclic redundancy check (CRC) generators, CRC checkers, multiple encoding and decoding 4b/5b, 8b/10b, 64b/66b, settable scramblers, various alignment and daisy-chaining options, and clock configurable front and backends.

The ability to loop the SERDES on itself at various stages is also very common. There are many commercially-available SERDES. Figure 3-3 and Figure 3-4 show example block diagrams.

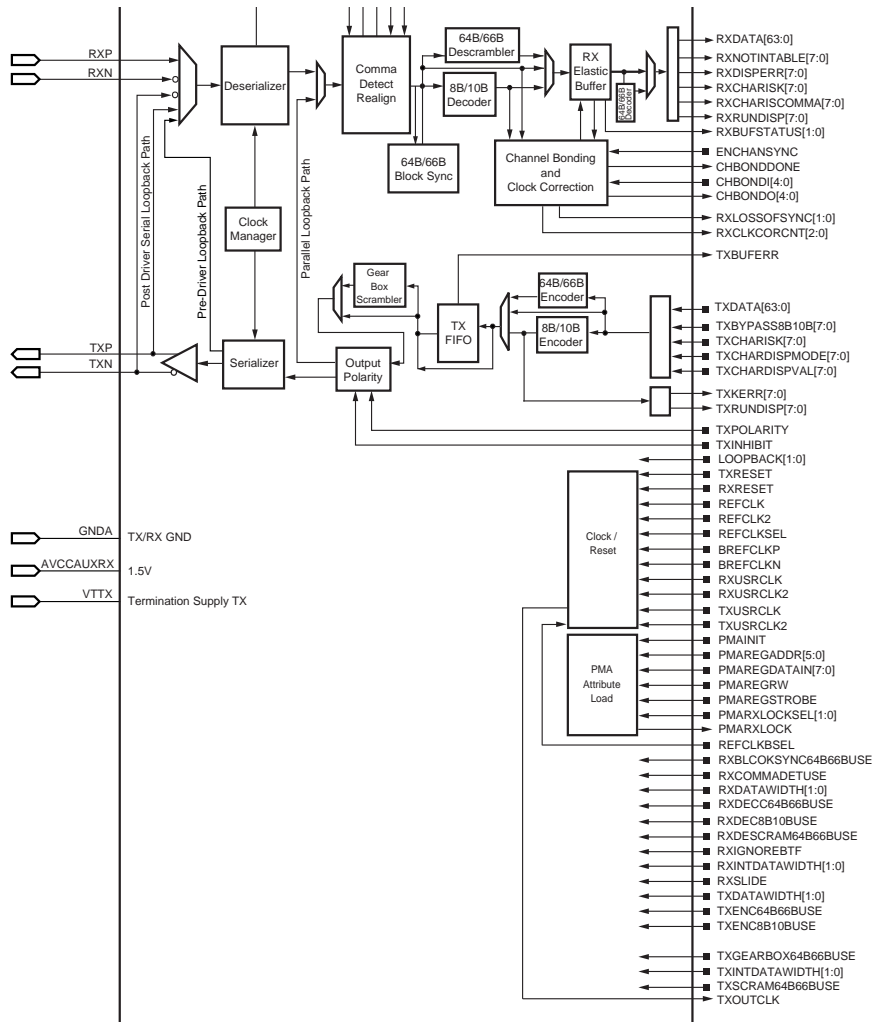


FIGURE 3-3: Virtex™-II Pro X RocketIO™ Block Diagram

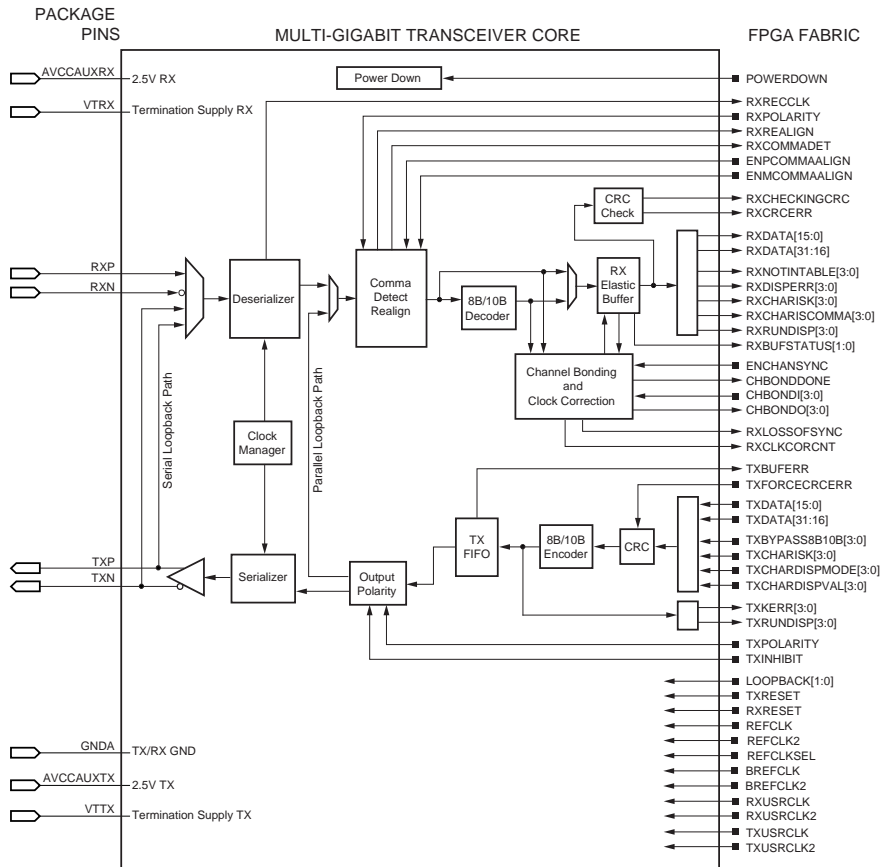


FIGURE 3-4: Virtex-II Pro RocketIO Block Diagram

Why Are They So Fast?

An unsettling aspect of the Gigabit SERDES is that they appear to be almost magical. They work with 3, 5, and even 10+ gigabits. How is that kind of speed possible? There are several techniques that provide this speed.

A common element of most of these techniques is multiple phases (Figure 3-5 and Figure 3-6). We can get an idea of how multiple phases can help us by looking at a multiphase data extraction circuit. If we have an incoming serial stream with a bit rate of x , we can recover the stream with a clock of $x/4$ by using multiple phases of the slow clock. The incoming stream is directed into four flip-flops, each running off a different phase of the clock (0, 90, 180, and 270).

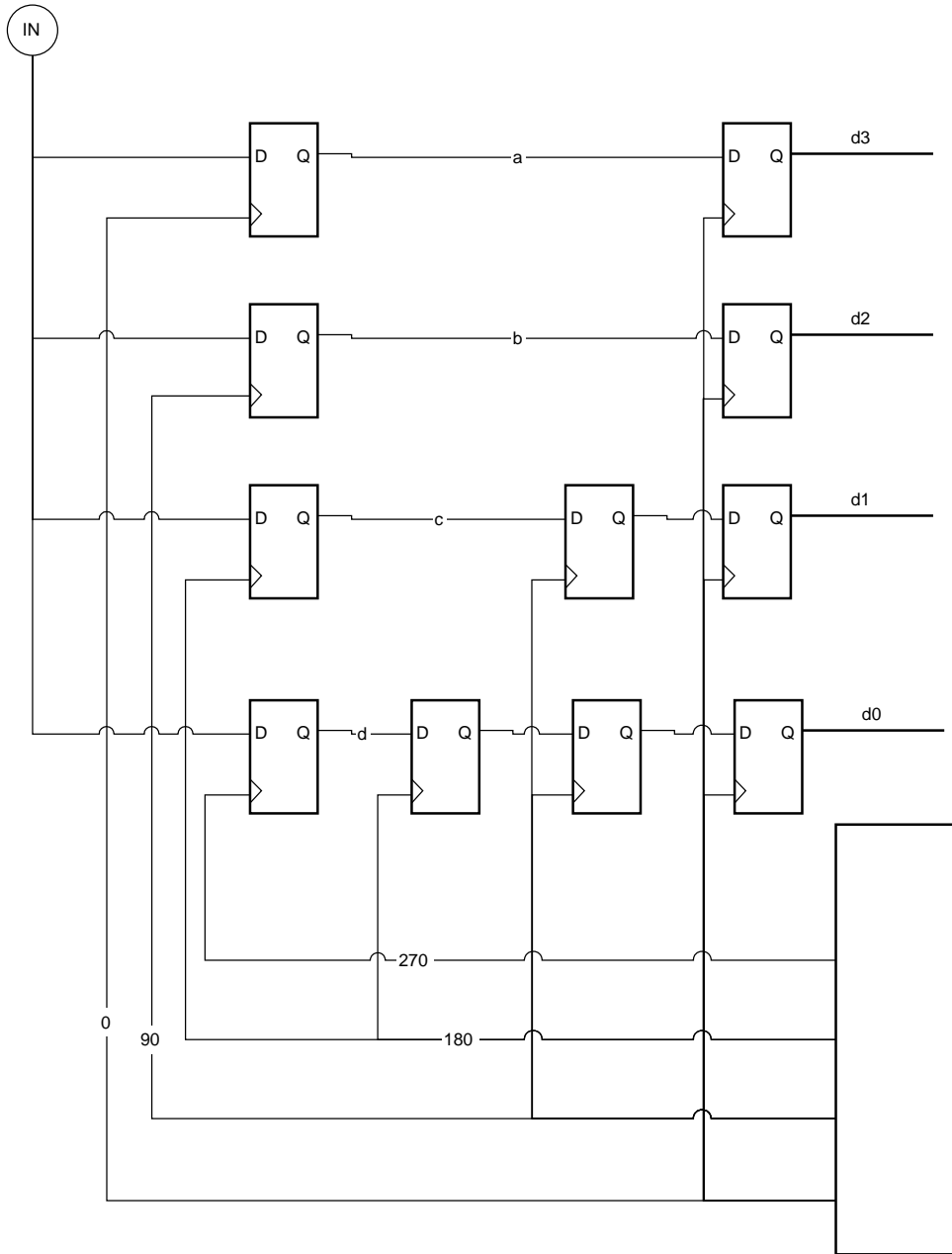


FIGURE 3-5: Multiphase Data Extraction Circuit

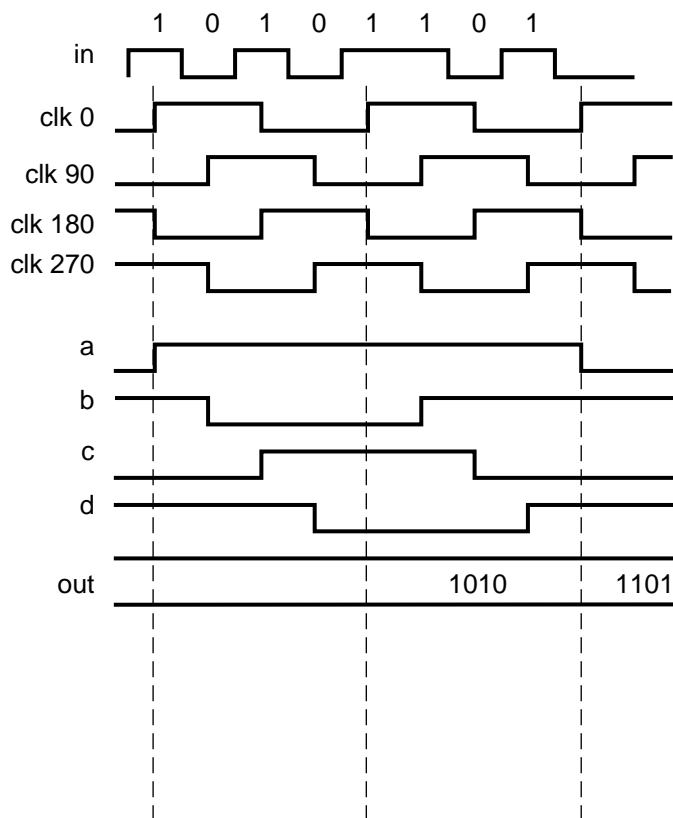


FIGURE 3-6: Example Waveform Multiphase Data Extraction Circuit

Each flip-flop then feeds into a flip-flop clocked by the next lowest phase until it is clocked off the zero-phase clock. This deserializes the incoming stream into a 4-bit word running at 1/4 the clock rate of the incoming stream.

In the previous example the phase was lined up and the clock was exactly 1/4 the rate of the incoming stream. How does that happen? We must lock to the incoming stream. We could do it with a classic phase-locked loop (PLL), but that would require a full-rate clock and defeat the purpose. One of the biggest advances in high-speed SERDES involves the PLLs used in clock and data recovery. A normal PLL requires a clock running at the data speed, but there are several techniques that can be used to avoid this requirement, including fractional rate phase detectors, multi-phase PLLs, parallel sampling, and over-sampling data recovery.

Line Encoding Schemes

Line encoding schemes modify raw data into a form that the receiver can accept. Specifically, the line encode scheme ensures that there are enough transitions for the clock recovery circuit to operate. They provide a means of aligning the data into words with a good direct current (DC) balance on the line.

Optionally, the line encoding scheme may also provide for implementation of clock correction, block synchronization and channel bonding, and division of the bandwidth into sub-channels.

There are two main line encoding schemes—value lookup schemes and self-modifying streams, or scramblers.

8b/10b Encoding/Decoding

The 8b/10b encoding scheme was developed by IBM and has been widely adapted. It is the encoding scheme used in Infiniband, Gigabit Ethernet, FiberChannel, and the XAUI interface to 10 Gigabit Ethernet. It is a value lookup-type encoding scheme where 8-bit words are translated into 10-bit symbols. These symbols ensure a good number of transitions for the clock recovery. Table 3-1 gives a few examples of 8-bit values that would result in long runs without transitions. 8b/10b allows for 12 special characters that decode into 12 control characters commonly called K-characters. We will look at K-characters in more detail, but first let's examine how 8b/10b ensures a good DC balance.

TABLE 3-1: Example of 8-bit Values

8-bit Value	10-bit Symbol
00000000	1001110100
00000001	0111010100

Running Disparity

DC balance is achieved in the 8b/10b through a method called running disparity. The easiest way to achieve DC balance would be to only allow symbols that have the same number of ones and zeros, but that would limit the number of symbols.

Instead, 8b/10b uses two different symbols assigned to each data value. In most cases, one of the symbols has six zeros and four ones, and the other has four zeros and six ones. The total number of ones and zeros is monitored and the next symbol is chosen based on what is needed to bring the DC balance back in line. The two symbols are normally referred to as + and - symbols. Symbol examples are given in Table 3-2.

TABLE 3-2: Examples of 8b/10b Symbols

Name	Hex	8 Bits	RD -	RD +
D10.7	EA	11101010	0101011110	0101010001
D31.7	FF	11111111	1010110001	0101001110
D4.5	A4	10100100	1101011010	0010101010
D0.0	00	00000000	1001110100	0110001011
D23.0	17	00010111	1110100100	0001011011

One additional benefit of the running disparity is that the receiver can monitor the running disparity and detect that an error has occurred in the incoming stream because the running disparity rules have been violated.

Control Characters

Table 3-3 lists the encoding of 12 special symbols known as control characters or K-characters.

TABLE 3-3: Valid Control K-Characters

Name	Hex	8 Bits	RD -	RD +
K28.0	1C	00011100	0011110100	1100001011
K28.1	3C	00111100	0011111001	1100000110
K28.2	5C	01011100	0011110101	1100001010
K28.3	7C	01111100	0011110011	1100001100
K28.4	9C	10011100	0011110010	1100001101
K28.5	BC	10111100	0011111010	1100000101
K28.6	DC	11011100	0011110110	1100001001
K28.7	FC	11111100	0011111000	1100000111
K23.7	F7	11110111	1110101000	0001010111
K27.7	FB	11111011	1101101000	0010010111
K29.7	FD	11111101	1011101000	0100010111
K30.7	FE	11111110	0111101000	1000010111

These control characters are used for alignment, control, and dividing the bandwidth into sub-channels.

Comma Detection

Alignment of data is an important function of the deserializer. Figure 3-7 represents valid 8b/10b data in a serial stream.

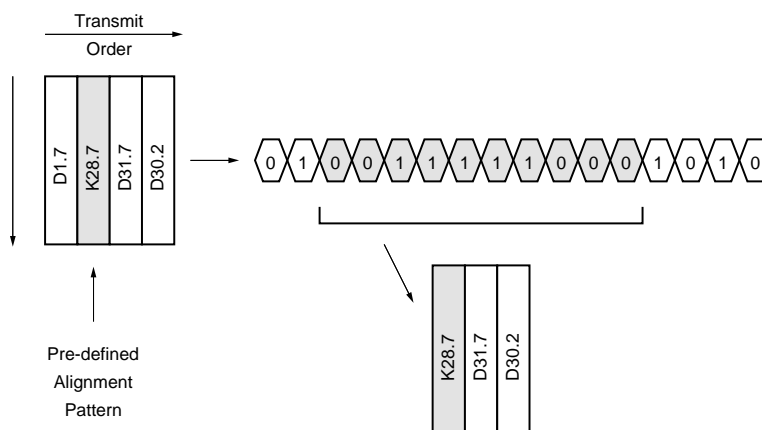


FIGURE 3-7: Serial Stream Valid 8b/10b Data

How do we know where the symbol boundaries are? Symbols are delineated by a comma. Here, a comma is one or two symbols specified to be the comma or alignment sequence. This sequence is usually settable in the transceiver, but in some cases it may be predefined.

Comma: One or two symbols specified to be the alignment sequence.

The receiver scans the incoming data stream for the specified bit sequence. If it finds the sequence, the deserializer resets the word boundaries to match the detected comma sequence. This is a continuous scan. Once the alignment has been made, all subsequent commas detected should find the alignment already set. Of course, the comma sequence must be unique within any combination of sequences.

For example, if we are using a signal symbol *c* for the comma, then we must be certain that no ordered set of symbols *xy* contains the bit sequence *c*. Using a predefined protocol is not a problem since the comma characters have already been defined.

One or more of a special subset of K-characters is often used. The subset consists of K28.1, K28.5, and K28.7, all of which have 1100000 as the first seven bits. This pattern is only found in these characters; no ordered set of data and no other K-characters will ever contain this sequence. Hence, it is ideal for alignment use. In cases where a custom protocol is built, the safest and most common solution is to “borrow” a sequence from a well-known protocol. Gigabit Ethernet uses K28.5 as its comma. Because of this it is often referred to as the comma symbol even though there are technically other choices.

The names used—such as D0.3 and K28.5—are derived from the way the encoders and decoders can be built. Figure 3-8 represents this method.

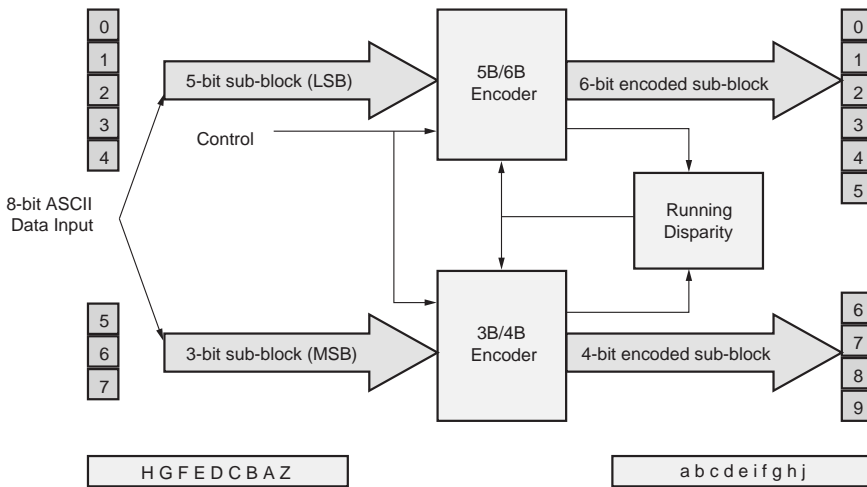


FIGURE 3-8: Encoders/Decoders Block Diagram

The 8-input bits are broken into 5- and 3-bit buses; that is how the names were developed. For example, the name *Dx.y* describes the data symbol for the input byte where the five least significant bits have a decimal value of *x* and the three most significant bits have a decimal value of *y*.

A *K* indicates the control character. The three bits turn into four bits and the five bits turn into six. Another naming convention refers to the 8-bit bits as *HGF EDCBA* and 10-bit bits as *abcdei fgbj*.

Overhead is one of the drawbacks to the 8b/10b scheme. To get 2.5 gigabits of bandwidth requires a wire speed of 3.125 Gb/s. Scrambling techniques can easily handle the clock transition and DC bias problems without a need for increased bandwidth.

Scrambling

Scrambling is a way of reordering or encoding the data so that it appears to be random, but it can still be unscrambled. We want randomizers that break up long runs of zeros and ones. Obviously, we want the descrambler to unscramble the bits without requiring any special alignment information. This characteristic is called a self-synchronizing code.

Scrambling: A way of reordering or encoding the data so that it appears random, but can be unscrambled.

A simple scrambler consists of a series of flip-flops arranged to shift the data stream. Most of the flip-flops simply feed the next bit, but occasionally a flip-flop will be exclusively ORed or ANDed with an older bit in the stream. Figure 3-9 shows this concept.

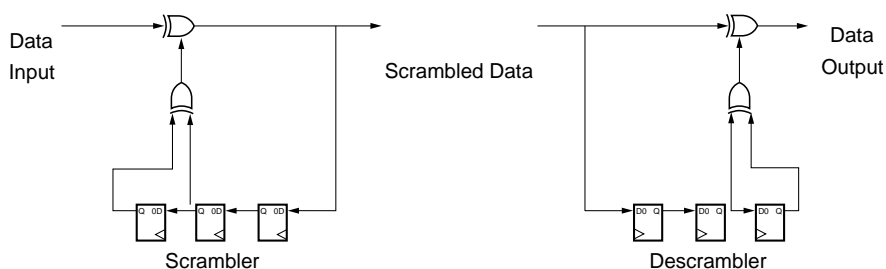


FIGURE 3-9: Basic Scrambling Circuit

The scrambling method is usually referred to as a polynomial because of the mathematics involved. Polynomials are chosen based on scrambling properties such as how random a stream they create, and how well they break up long runs of zeros and ones. They must also avoid generating long run lengths.

Increasing the clock rate of the flip-flops is desirable. But obtaining a high rate such as 10 Gb/s is simply not attainable. However, there is a way to parallel any serial coefficient into a *y*-size parallel word to speed up the process as shown in Figure 3-10.

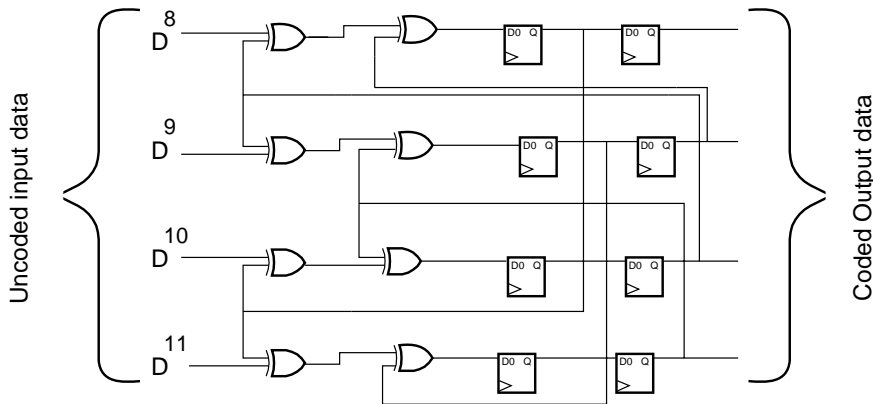
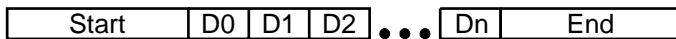


FIGURE 3-10: Parallel Scrambling Circuit

Scrambling eliminates long runs and works to eliminate other patterns that may have a negative impact on the receiver’s ability to decode the signal. There are, however, other tasks provided by line encoding schemes such as 8b/10b that are not supplied by scrambling:

- Word alignment
- Clock correction mechanism
- Channel bonding mechanism
- Sub-channel creation

While the last three may not be needed in some circumstances, word alignment is always needed. If scrambling is used as the line encoding method, then another method must be used for word alignment. For example, we can exclude some values from the allowed values of the data or the payload. Then we can use these disallowed values to create a stream of bits that could not occur in the data portion of the sequence (Figure 3-11).



$Sart = 36 \text{ bits} = 3F \ 00 \ 00$
 $End = 36 \text{ bits} = 3F \ 3F \ 00$
 $Data = 12 \text{ bits range is } 04 - 3B$

00,01,02,03,3c,3d,3e,3f,
 are forbidden in Data fields.

FIGURE 3-11: Data Frame Designed for Scrambling

Normally, this would involve designing long run lengths that cannot occur in the data stream because of the disallowed values. The long runs will be broken by the scrambling and then restored when the stream is unscrambled. Downstream unscrambler logic looks for these patterns and aligns the data. Similar techniques can be used to install any of the other characteristics.

4b/5b 64b/66b

4b/5b is similar to 8b/10b, but simpler. As the name implies, four bits are encoded into five bits with this scheme. 4b/5b offers simpler encoders and decoders than 8b/10b. But there are few control characters and it does not handle the DC balance or disparity problem. With the same coding overhead and less functionality, 4b/5b is not often used anymore. Its main advantage was implementation size, but gates are so cheap now that it is not much of an advantage. 4b/5b is still used in various standards including low bit rate versions of FiberChannel and Audio Engineering Society-10 (AES-10) or Multichannel Audio Digital Interface (MADI), a digital audio multiplexing standard.

One of the new encoding methods is known as 64b/66b. We might think that it is simply a version of 8b/10b that has less coding overhead, but the details are vastly different.

64b/66b came about as a result of user needs not being met by current technology. The 10 Gigabit Ethernet community had a need for Ethernet-based communication at 10 Gb/s. And while they could use four links at a 2.5 Gb payload and 3.125-Gb/s wire speed, XERDES was approaching the ultimate 10 Gb solution in a single link. There were new SERDES that could run at just over 10 Gb/s, but could not be pushed to the 12.5 Gb needed to support 8b/10b overhead.

The laser driving diode was another issue. The telecommunications standard Synchronous Optical Network (SONET) used lasers capable of just over 10 Gb. Faster lasers were much more expensive. The Gigabit Ethernet community could either give up or create something with a significantly lower overhead to replace 8b/10b. They chose 64b/66b.

64b/66b: A line encoding scheme developed for 10 Gigabit Ethernet that uses a scrambling method combined with a non-scrambled sync pattern and control type.

Rather than using a 8b/10b-type lookup table, 64b/66b uses a scrambling method combined with a non-scrambled sync pattern and control type. Figure 3-12 illustrates the 64b/66b scheme.

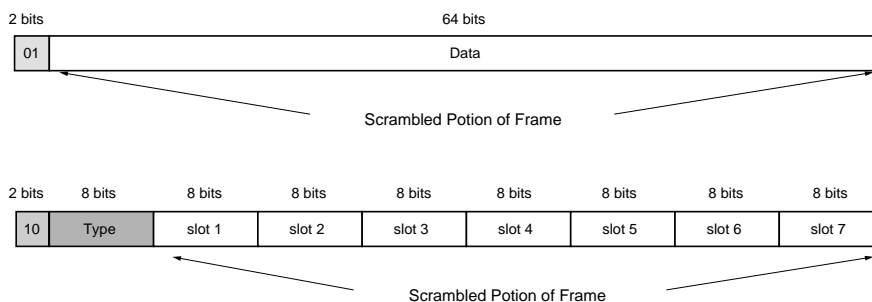


FIGURE 3-12: 64b/66b Diagram

There are two main frame types. The simple main frame consists of a 2-bit sync pattern of 01 followed by 64 bits of data. The data is scrambled but the sync bits are not. The other frame type allows for control information as well as data. Control frames start with the 2-bit pattern 10. The eight bits in the type field define the format of the 56-bit payload. For example, if the type is hex *0xcc*, then the pattern contains four bytes of data and three bytes of control (Figure 3-13).

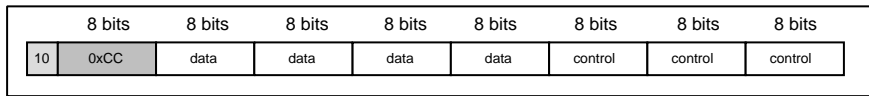


FIGURE 3-13: 0xcc Type Example

There is also a zero-bit wide symbol associated with this frame (Figure 3-14).

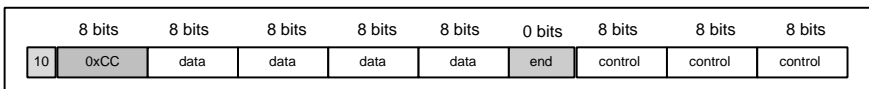


FIGURE 3-14: 0xcc Type with Symbol Shown

How is a zero-bit wide symbol created? It is not actually 0 bits; it is part of the type byte that is projected into the payload. There are eight bits for the type field that would allow for 256 different types of payloads. Most 64b/66b systems define about 15 different types. And those 15 types simply define *x* data bytes followed by *y* control bit. They may also include a reversal *x* control then *y* data bytes. It is common to define the placement of these inferred symbols with some types. Common zero-

bit wide symbols are t (*end*) and s (*start*). A complete list of control block formats for one 64b/66b implementation is given in Figure 3-15.

Input Data	S y n c	Block Payload								
Bit Position	01	2							65	
Data Block Format	01	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	
Control Block Formats		Block Type Field								
C ₀ C ₁ C ₂ C ₃ C ₄ C ₅ C ₆ C ₇	10	0x1e	C ₀	C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇
C ₀ C ₁ C ₂ C ₃ O ₄ D ₅ D ₆ D ₇	10	0x2d	C ₀	C ₁	C ₂	C ₃	O ₄	D ₅	D ₆	D ₇
C ₀ C ₁ C ₂ C ₃ S ₄ D ₅ D ₆ D ₇	10	0x33	C ₀	C ₁	C ₂	C ₃		D ₅	D ₆	D ₇
O ₀ D ₁ D ₂ D ₃ S ₄ D ₅ D ₆ D ₇	10	0x66	D ₁	D ₂	D ₃	O ₀		D ₅	D ₆	D ₇
O ₀ D ₁ D ₂ D ₃ O ₄ D ₅ D ₆ D ₇	10	0x55	D ₁	D ₂	D ₃	O ₀	O ₄	D ₅	D ₆	D ₇
S ₀ D ₁ D ₂ D ₃ D ₄ D ₅ D ₆ D ₇	10	0x78	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	
O ₀ D ₁ D ₂ D ₃ C ₄ C ₅ C ₆ C ₇	10	0x4b	D ₁	D ₂	D ₃	O ₀	C ₄	C ₅	C ₆	C ₇
T ₀ C ₁ C ₂ C ₃ C ₄ C ₅ C ₆ C ₇	10	0x87		C ₁	C ₂	C ₃	C ₄	C ₅	C ₆	C ₇
D ₀ T ₁ C ₂ C ₃ C ₄ C ₅ C ₆ C ₇	10	0x99	D ₀		C ₂	C ₃	C ₄	C ₅	C ₆	C ₇
D ₀ D ₁ T ₂ C ₃ C ₄ C ₅ C ₆ C ₇	10	0xaa	D ₀	D ₁		C ₃	C ₄	C ₅	C ₆	C ₇
D ₀ D ₁ D ₂ T ₃ C ₄ C ₅ C ₆ C ₇	10	0xb4	D ₀	D ₁	D ₂		C ₄	C ₅	C ₆	C ₇
D ₀ D ₁ D ₂ D ₃ T ₄ C ₅ C ₆ C ₇	10	0xcc	D ₀	D ₁	D ₂	D ₃		C ₅	C ₆	C ₇
D ₀ D ₁ D ₂ D ₃ D ₄ T ₅ C ₆ C ₇	10	0xd2	D ₀	D ₁	D ₂	D ₃	D ₄		C ₆	C ₇
D ₀ D ₁ D ₂ D ₃ D ₄ D ₅ T ₆ C ₇	10	0xe1	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅		C ₇
D ₀ D ₁ D ₂ D ₃ D ₄ D ₅ D ₆ T ₇	10	0xff	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	

FIGURE 3-15: Control Block Formats for an Example 64b/66b Implementation

Careful observation of the table reveals the O symbol. This symbol is used to define ordered sets. This allows a protocol that was supposed to be used with 8b/10b to be converted to use 64b/66b.

Now let's examine a line encoding scheme. We will examine each of the main functions of a line encoding scheme and see how they are accomplished.

Sufficient Transitions

Scrambling of the payload section will provide adequate transition for clock recovery. Careful selection of the scramblers will also handle DC bias problems. The scrambler used in 64b/66b is $X_{58} + X_{19} + 1$.

Alignment

64b/66b differs from other methods in the alignment procedure. Figure 3-16 shows how it works.

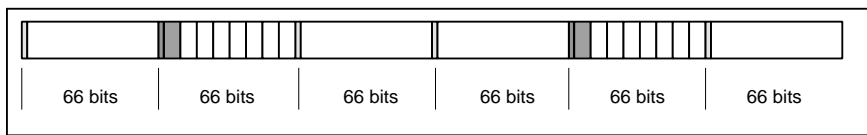


FIGURE 3-16: 64b/66b Alignment Procedure

There will be a sync value of *01* or *10* every 66 bits. Those same bit combinations will appear in many other places as well. The alignment procedure selects a random starting point. It first looks for a valid sync (*01* or *10* combination). If there isn't one, it slips a bit and rechecks. Once a *01* or *10* combination is found, the position 66 bits later is checked. If that is a valid sync also, the process increments the counter and checks the location 66 bits later. If enough sync markers are found in a row without any misses, the alignment is considered found. Any misses during the sequence forces the counter back to zero.

Once the alignment has been locked, missed syncs are considered errors. If enough errors occur in a period of time, the alignment is re-evaluated. At first glance, it appears that this algorithm would obtain lock within the maximum number of valid sync tries (+66 or less). But the high likelihood of the *01* and *10* sequences showing up in the data window can mean many false paths are taken for a long time before they are abandoned.

To speed lock time, some optional or alternative protocols have been suggested. They involve the replacement of data with special training or locking sequences that can ease alignment.

Clock Correction

Clock correction can be handled on byte or multi-byte boundaries, or on a 66-bit code word. A special type could be defined to be the clock correction symbol. The entire payload of the code word would not contain useful information and could be deleted or repeated as necessary. Alternatively, a byte-wide clock correction symbol could be defined as any unused value. Of course, if the SERDES we are using only supports one of these methods, we will need to use that particular method. The byte-wide method is the most common since it allows for smaller receive FIFO buffers and matches up better with legacy protocols.

Channel alignment

Channel alignment can be handled much like clock alignment, either as a special type or a sequence found within the control data.

Sub-Channels

Sub-channels can be handled like clock alignment, either as a special type or a sequence found within the control data.

4b/5b 64b/66b Trade-Offs

These functions comprise the overhead coding method that allowed 10 Gigabit Ethernet to use existing SONET class laser diodes. Laser diodes are not the only similarity that this method has in common with SONET; SONET uses many of the same principles for alignment but is even more complicated than 64b/66b.

The price for the lower overhead is longer alignment times, the possibility of a slight DC bias, and more complicated encoders and decoders. Complications such as turning the scramblers on and off for payload vs. sync and type fields make 64b/66b circuits more complicated than their 8b/10b cousins. There is also a complexity cost for using and setting up the encoder.

Introduction to Packets

Some designers feel that sending data over packets for anything but a local area network (LAN) is a complete waste. Let's address that issue by first defining a packet.

Packet: A well-defined collection of bytes consisting of a header, data, and trailer.

Notice that there is nothing in the definition about source and destination addresses, CRCs, minimum lengths, or Open Systems Interconnection (OSI) protocol layers. A packet is simply a data structure with defined starting and ending points. While LAN packets often have many of these characteristics, there are many other uses of packets that are much simpler.

Packets are used everywhere to transfer information—automobile wiring harnesses, cell phones, and home entertainment centers, to name a few. But what do packets have to do with gigabit serial links?

Most data transferred across a gigabit serial link is embedded in some sort of packet. It's only natural that a SERDES requires a method for aligning the incoming stream into words. This special bit sequence or comma must be sent if the system requires clock correction. The comma could be a natural marker for the beginning or end of a frame. If clock correction is required, the clock correction sequence is usually the ideal character. After adding a couple of ordered sets to indicate the end or start of the packet, and an ordered set to indicate a special type of packet, we have a simple, powerful transmission path.

The idle symbol, or sequence, is another important packet concept. This symbol is sent whenever there is no information to send. Continuous transmission of data ensures that the link stays aligned

and that the PLL keeps the recovered clock locked. Figure 3-17 illustrates some sample packet formats from various standards.

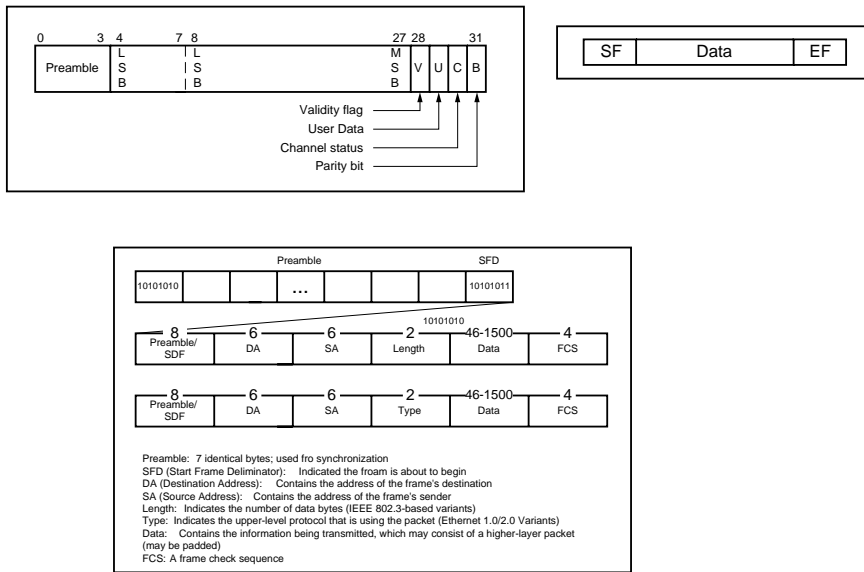


FIGURE 3-17: Packet Format Diagrams

Reference Clocking Requirements

The input, or reference clock, of a Multi-Gigabit Transceiver (MGT) has very tight specifications. It includes a tight frequency requirement usually specified in allowable parts per million (PPM) of frequency error. It will also have strict jitter requirements defined in terms of time units (picoseconds) or unit intervals (UI).

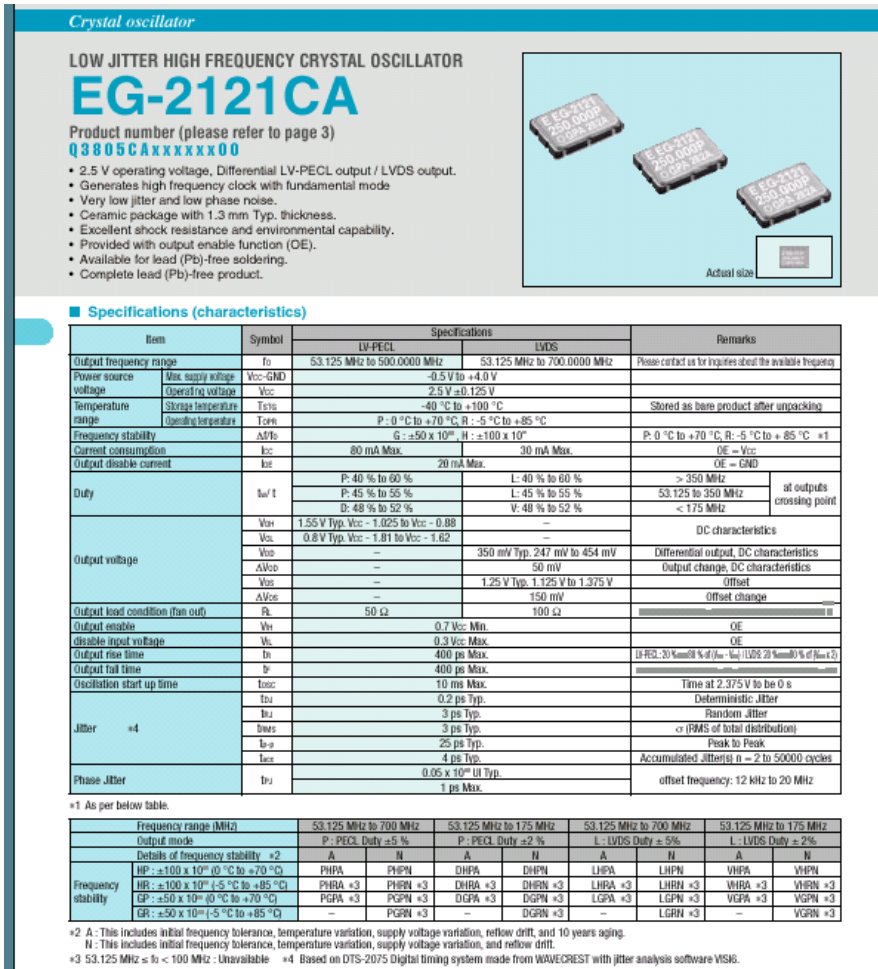
PPM: Parts per million; a way of describing a very small ratio.

UI: Unit intervals; same as length of time as a symbol, i.e., $0.2 \text{ UI} = 20\%$ of the symbol time.

Jitter: Variation of the ideal transition placement.

Such tight requirements enable the PLL and clock extraction circuits to work. This often requires an accurate crystal oscillator on each printed circuit board (PCB) in the system that uses MGTs. These crystal oscillators are a step above most used for digital systems and will cost more. In many cases,

clock generation chips and PLLs have too much jitter to be used. Figure 3-18 shows some recently-announced clock generation units that are good enough to work with multi-gigabit SERDES.



K-characters and/or data characters as the clock correction sequence. In some cases a four-symbol clock correction sequence may be desired. Clock correction works by monitoring the receive FIFO. If the FIFO is getting close to full, it simply looks for the next clock correction sequence and does not write that data sequence into the FIFO. This is called *dropped*. Conversely, if the FIFO is getting close to empty, the next time a clock correction sequence is found it will be written into the FIFO twice. This is commonly referred to as *repeating*.

The clock correction must happen often enough to allow dropping or repeating to compensate for the differences in the clocks. Often the clock correction sequence will also be the same as the idle sequence.

Some systems do not require clock correction. In many chip-to-chip applications, for example, the same oscillator will provide the reference clock to all transmitters. Using the same reference clock and same rate means there is no need for clock correction. Also, a clock correction is not needed when all of the receive circuitry is clocked from the recovered clock. If the FIFO is emptied at the same rate it is filled, there is no need for clock correction.

Also, clock correction is not required when all transmit reference clocks are locked using an external PLL to a common reference. This is a common architecture for high definition serial digital video links. All transmit clocks are derived from a common video reference. Failure to lock to this signal will usually result in a free running video stream that tends to *roll* in respect to the rest of the locked signals. While achieving this at one or two gigabits is easily possible, designing PLLs with enough accuracy to provide the input reference clocks for 10-Gb links is quite challenging.

Table 3-4 shows the maximum number of clocks between clock correction sequences and their accuracy for various oscillator frequencies.

TABLE 3-4: Clock Correction Table

Oscillator Frequency (MHz)	OSC Accuracy (PPM)	Line Speed (GB/s)	Fmax (MHz)	Fmin (MHz)	Diff/Cycle (ps)	Max Cycles before Correction			
						Remove 1 Sequence	Remove 2 Sequences	Remove 3 Sequences	Remove 4 Sequences
156.25	100	3.125	156.2656	156.2344	1.2800	4,999	9,999	14,998	19,998
156.25	50	3.125	156.2578	156.2422	0.6400	9,999	19,998	29,998	'39,997
156.25	20	3.125	156.2531	156.2469	0.2560	24,999	49,999	74,998	99,998
125	100	2.500	125.0125	124.9875	1.6000	4,999	9,998	14,998	19,997
125	50	2.500	125.0063	124.9938	0.8000	9,999	19,998	29,998	'39,997
125	20	2.500	125.0025	124.9975	0.3200	24,999	49,998	74,998	99,997
62.5	100	1.250	62.5063	62.4938	3.2000	4,999	9,998	14,998	19,997
62.5	50	1.250	62.5031	62.4969	1.6000	9,999	19,998	29,998	'39,997
62.5	20	1.250	62.5013	62.4988	0.6400	24,999	49,998	74,998	99,997

Receive and Transmit Buffers

The receive and transmit buffers, or FIFOs, are the main digital interface of the Multi-Gigabit Transceiver. This is normally where data is written and read. On the transmit side it is common to have a small FIFO that requires the read and the write clock to be isochronous (matched in frequency but not necessarily matched in phase).

Isochronous: Matched in frequency but not necessarily matched in phase.

A different scheme is used in cases where the tx_write and tx_read strobes are not of the exact same frequency. Here, a larger FIFO is used and its current status is constantly monitored. If the FIFO is filling it will eventually overrun. In this case the incoming stream is monitored for *idle* symbols. When encountered they are not written into the FIFO.

Conversely, if the FIFO is running low when an idle is found on the output, the data is brought to the user. The write pointer is not moved causing the idle to be repeated. It is important for idle symbols to be used instead of byte alignment, comma symbols, clock correction sequences, or channel bonding sequences. All these are needed downstream at some guaranteed delivery rate.

The receive FIFO built into an MGT is usually considerably deeper than the transmit (Tx) buffer. Its main purpose is to allow for clock correction and channel bonding.

Channel Bonding

Sometimes there is a need to move more data than can fit on one serial link. In these cases multiple links are used in parallel to transmit the data. When this is done, incoming streams must be *aligned*. This process is commonly referred to as channel bonding. Channel bonding absorbs the skew between two or more MGTs and presents the data to the user as if it were transmitted over a single link (Figure 3-19).

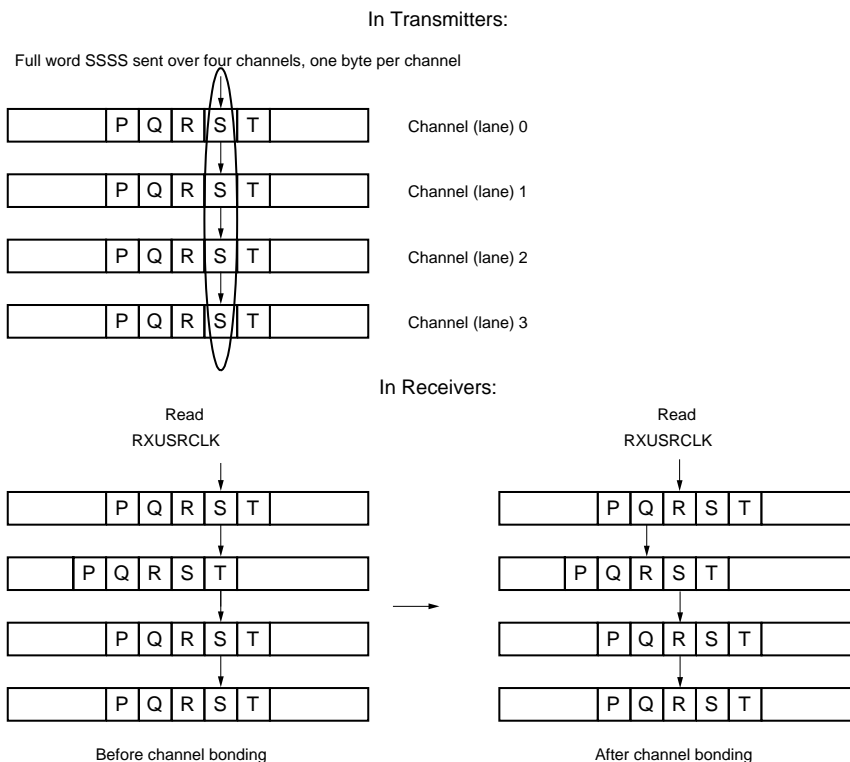


FIGURE 3-19: Channel Bonding Block Diagram

Channel Bonding: Absorbs the skew between two or more MGTs and presents the data to the user as if it were transmitted over a single link.

There are several causes of data skew between multiple MGTs:

- Differences in transmission path length
- Active repeaters in transmission path
- Differences because of clock correction
- Differences in time to lock/byte alignment

Since channel bonding requires communication between transceivers, the exact details will vary from vendor to vendor and part to part. Some common traits are designation of one channel as the master channel, designation of slaves, and possibly the designation of forwarding slaves. Three-level channel bonding that includes a master and forwarding slaves is sometimes referred to as two-hop channel bonding.

The channel bonding sequence must be unique and expandable and it must be ignored downstream because it may be added or dropped. There are normally a minimum number of symbols between a clock correction sequence and a channel bonding sequence. Many 8b/10b-based standard protocols specify a minimum of four symbols between clock correction and channel bonding sequences. Hence, four symbols or bytes is a common separation distance.

Physical Signaling

The physical implementation of multi-gigabit SERDES universally takes the form of differential-based electrical interfaces. There are three common differential signal methods—Low-Voltage Differential Signaling (LVDS), Low Voltage Pseudo Emitter-Coupled Logic (LVPECL), and Current Mode Logic (CML). CML is preferred for the gigabit link. It has the most common interface type and often provides for either AC or DC termination and selectable output drive. Some inputs provide built-in line equalization and/or internal termination. Often the termination impedance is selectable as well.

Front-end and back-end together make up the physical interface.

CML: Current Mode Logic; a differential-based electrical interface well suited to the gigabit link.

Figure 3-20 shows a CML-type driver. The concept behind this high-speed driver is quite simple. One of the two resistors always has a current running through it that is different than the current running through the other. Figure 3-21 illustrates an MGT receiver.

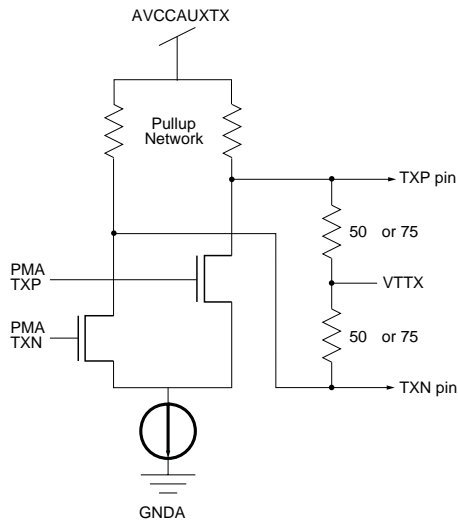


FIGURE 3-20: CML Driver

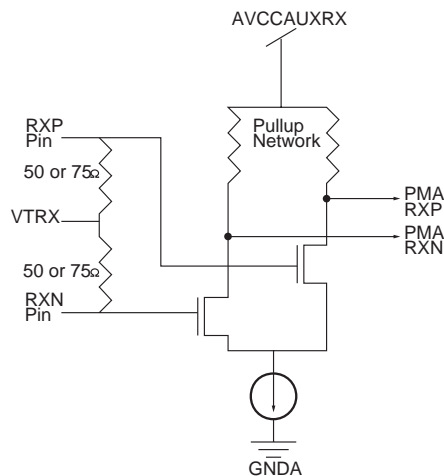


FIGURE 3-21: MGT Receiver

Figures 3-22 and Figure 3-23 list data sheets for the analog front-end and back-end.

Parameter		Min	Typ	Max	Units	Conditions
V_{IN}	Serial input differential peak to peak (RXP/RXN)	175		2000	mV	
V_{ICM}	Common mode input voltage range	500		2500	mV	
T_{ISKEW}	Differential input skew			75	ps	
T_{JTOL}	Receive data total jitter tolerance (peak to peak)			0.65	UI ⁽¹⁾	
T_{DJTOL}	Receive data deterministic jitter tolerance (peak to peak)			0.41	UI	

Notes:

1. UI – Unit Interval

FIGURE 3-22: Differential Receiver Parameters

Parameter		Min	Typ	Max	Units	Conditions
V_{OUT}	Serial output differential peak to peak (TXP/TXN)	800		1600	mV	Output differential voltage is programmable
V_{TTX}	Output termination voltage supply	1.8		2.625	V	
V_{TCM}	Common mode output voltage range (no transmission line connected)	1.1		1.5	V	
V_{TCM}	Common mode output voltage range (transmission line connected)	1.1		2.0	V	The common mode depends on coupling (DC or AC), VTTX, VTRX, and differential swing. Spice simulation gives the exact common mode voltage for any given system.
V_{ISKEW}	Differential output skew			15	ps	

FIGURE 3-23: Differential Transmitter Parameters

Pre-Emphasis

Perhaps the most important characteristic of a multi-gigabit driver is its ability to perform pre-emphasis. Pre-emphasis is the intentional overdriving at the beginning of a transition. To the inexperienced eye it looks like a fault; it looks like *overshoot* and *undershoot* that can indicate a bad design. To understand why this is done, we need to understand inter-symbol interference (ISI).

Pre-emphasis: Intentional overdriving at the first of a transition.

ISI occurs when the serial stream contains a number of bit times of the same value followed by short (1 or 2) bit times of the opposite value. The medium (transmission path capacitance) has less time to charge during the shorter value time, so it produces lower amplitude.

ISI: Inter-symbol interference— Occurs when the serial stream contains a number of bit times of the same value followed by short bit times of the opposite value.

With ISI, the larger runs allow for maximum charge but the single bit time cannot compensate. It is at risk of not being detected. Figure 3-24, Figure 3-25, and Figure 3-26 show this phenomena. The solution to this problem is to overdrive the first of each transition, or underdrive any consecutive bit times of the same value. This is sometimes called *de-emphasis*.

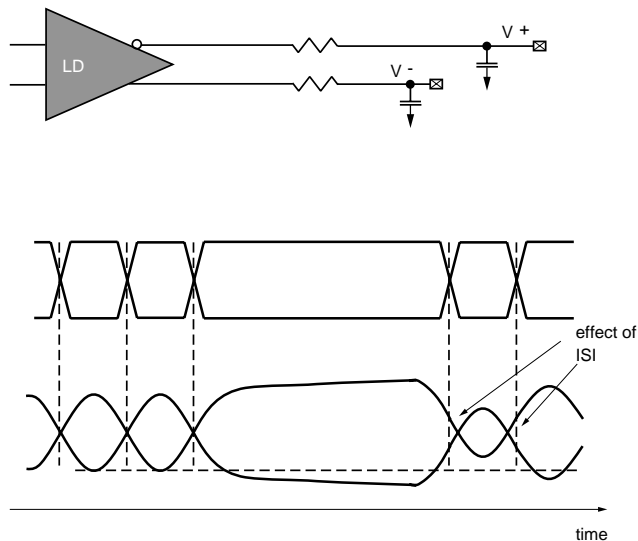


FIGURE 3-24: Inter-Symbol Interference

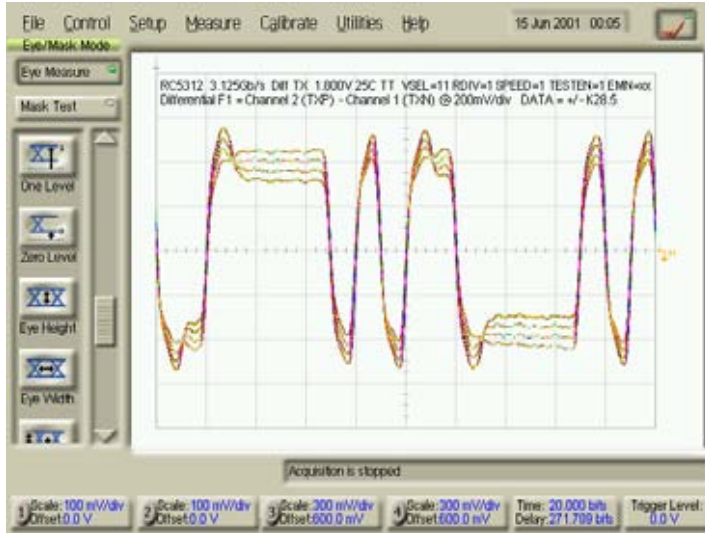


FIGURE 3-25: DCA Screen Capture

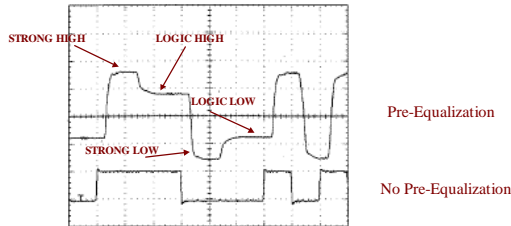
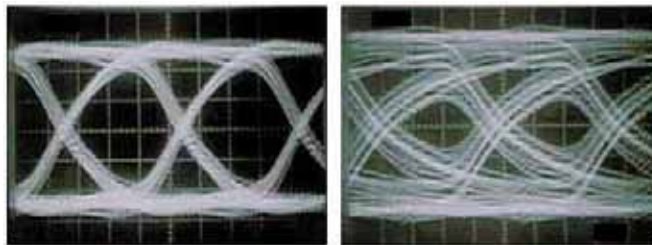


FIGURE 3-26: Pre-Emphasis

The two eye diagrams in Figure 3-27 show the improved eye opening that occurs when pre-emphasis is used to reduce ISI.



Pre-Emphasis

No Pre-Emphasis

FIGURE 3-27: Eye Diagrams with and without Pre-Emphasis

Eye pattern: Common waveform viewed on digital sampling scopes. It is an indication of the quality of the signal. Jitter, impedance matching, and amplitude can all be characterized through eye patterns.

Pre-emphasis can be implemented by using two CML drivers in parallel where one is delayed one bit time after the other. Figures 3-28 and 3-29 show a sample circuit and the waveforms that drive the transistors to obtain the output.

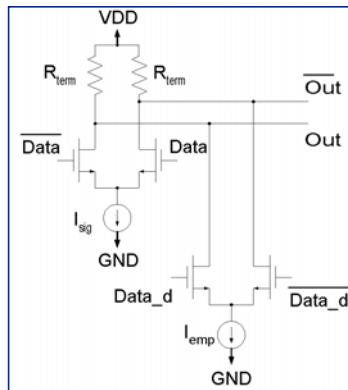


FIGURE 3-28: Pre-Emphasis CML Schematic

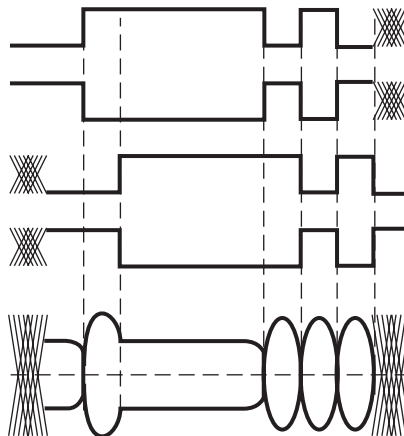


FIGURE 3-29: Timing Diagrams for CML Circuit in Figure 3-28

Differential Transmission Lines

Digital design engineers and PCB designers once thought of traces as simple interconnects or wires. In fact, prototypes were built using a technique called *wire warping*. Transmission lines and transmission line theory were not necessarily applied. When the propagation delay of the trace was a tiny fraction

of the rise time of the signal, this was satisfactory. But as signals increased in frequency, transmission line theory had to move into the PCB design process.

For multi-gigabit operation this includes not only transmission lines and controlled impedance, but differential pair controlled impedance as well.

Differential pair impedance matched traces are two traces that run adjacent to each other. The spacing between the pair allows for a coupling to occur between the traces. The coupling is called *weak* (Figure 3-30) if the traces are relatively far apart. If the traces are closer it is called *strong* (Figure 3-31).

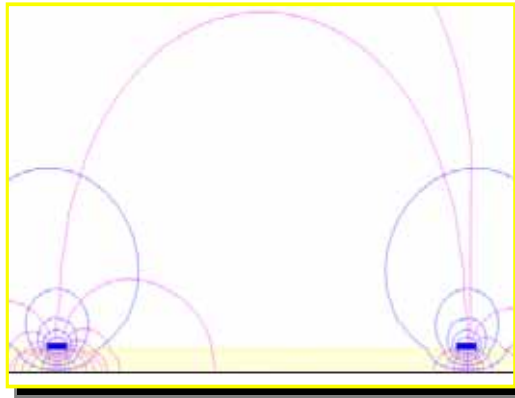


FIGURE 3-30: Weakly Coupled

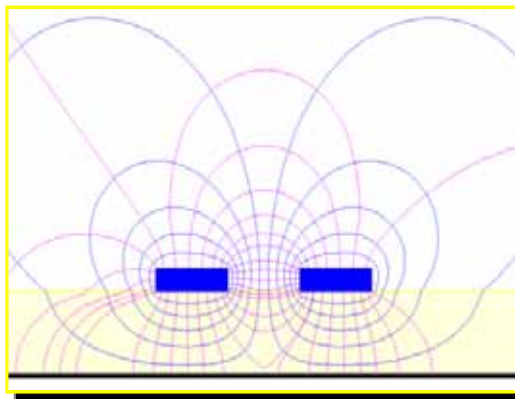


FIGURE 3-31: Strongly Coupled

The coupling also affects the impedance of the trace if a given trace length and layer stack will make a given impedance. The same geometry for a differential pair will have a different impedance. The exact dimensions for any given impedance varies on material, but the board manufacturer can often provide the exact dimensions.

A tool/mathematical model called a *field solver* can calculate the numbers as well. Figure 3-32 shows the main types of controlled impedance differential traces—microstrip, stripline, offset stripline, and broadside coupled. Table 3-5 includes controlled impedance differential trace types.

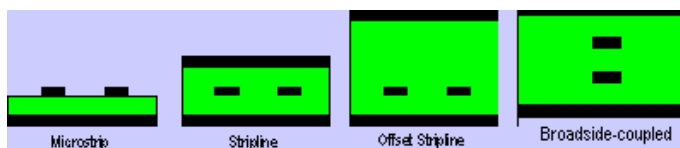


FIGURE 3-32: Controlled Impedance Differential Traces

TABLE 3-5: Types of Controlled Impedance Differential Traces

Type	Pros	Cons
Microstrip	Less loss than internal traces	Only two layers (top and bottom) More susceptible to interference
Stripline	Better shielding More possible layers	More amplitude loss per inch in high frequency signals than microstrip
Offset Stripline	Useful if non-symmetrical Stack-up is needed. Can be used to limit the number of power/gnd planes.	If used to save layers, the offset area above the traces should be kept free of other traces and must be free of parallel traces.
Broadside-coupled	Very tight coupling	The broadside coupled is difficult to manufacture because of tight tolerances and it is not recommended for multi-gigabit operation.

Line Equalization

Equalization is an attempt to compensate for differences in impedance/losses relative to frequency. Equalizers come in many forms but can generally be divided into passive and active types.

Active equalizer: Frequency dependant amplifiers/attenuators.

Passive equalizer: A passive circuit with a frequency response that is complementary to the transmission losses; similar to a filter.

A passive equalizer is a passive circuit that has a frequency response that is complementary to the transmission losses. A passive equalizer can be thought of as a filter. If we filter out the frequencies that

the transmission line passes, and not filter those that it does not pass, we can flatten the overall response as shown in Figure 3-33.

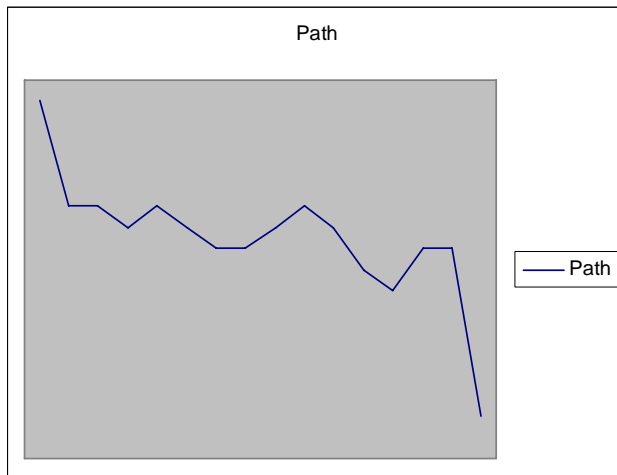


FIGURE 3-33: Unequalized System Frequency Response Example

The active equalizers can be thought of as frequency-dependant amplifiers/attenuators. There are two types of active equalizers—fixed pattern and self-adjusting. No matter what the incoming data stream looks like, the fixed pattern active equalizer has the same frequency response (Figure 3-34 and Figure 3-35).

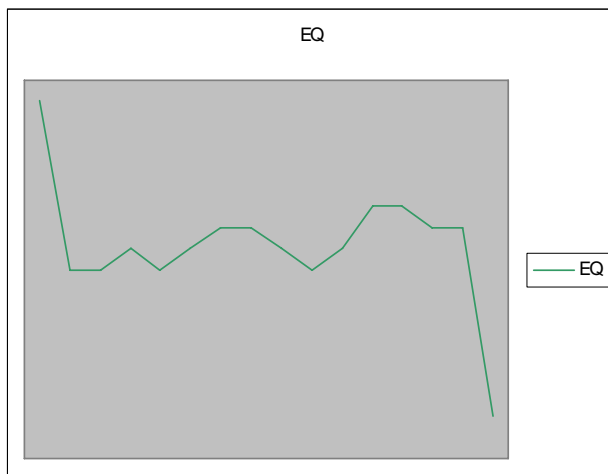


FIGURE 3-34: Equalizer Frequency Response Example

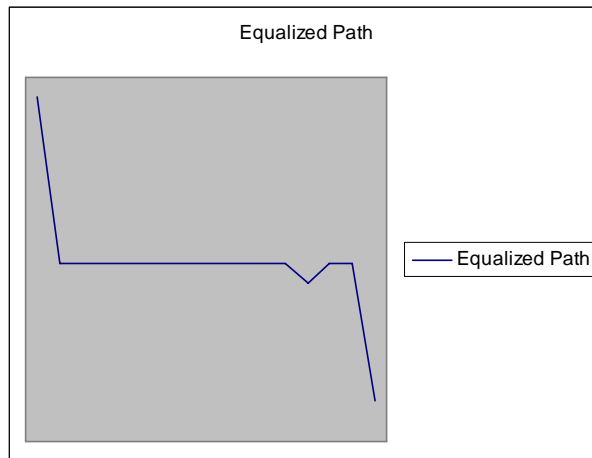


FIGURE 3-35: Equalized System Frequency Response Example

This set gain attenuation pattern may be user-selectable or programmable. Some have a simple control— n settings with high or low gain. They are similar to the bass control on a simple audio system. Or they could allow for individual settings at various frequency bands much like the equalizer settings on a more complex audio system. A chart showing the possible frequency response for one such equalizer is shown in Figure 3-36.

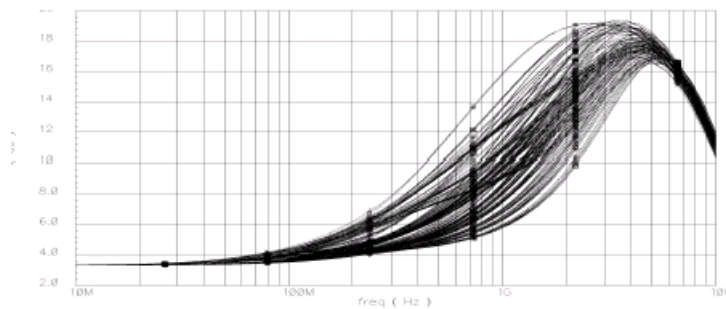


Figure 4-14: Magnitude (dB) vs. Frequency (Hz) Plot for all 1024 states of RXFER[9:0]

FIGURE 3-36: Sample Equalizer Frequency Response

The self-adjusting, or learning, equalizer is more complicated. It analyzes the incoming signal and detects which frequencies are being attenuated by the transmission path. Adjustments and measurements are made in a closed loop-type system. A self-adjusting equalizer is dependent on the incoming bit stream.

Fixed pattern and self-adjusting are the two types of active equalizers.

Often this type of equalizer is designed to work with a specific type of line encoding scheme. Learning equalizers are best suited for links with variable channels such as variable cable lengths or backplanes systems with significant differences in slot positions.

Fixed equalizers are better suited for systems with no variability such as chip-to-chip, balanced backplanes, and fixed-length cable systems. Equalizers are sometimes included in the analog front end of the SERDES or added to a system as a separate component (Figure 3-37).

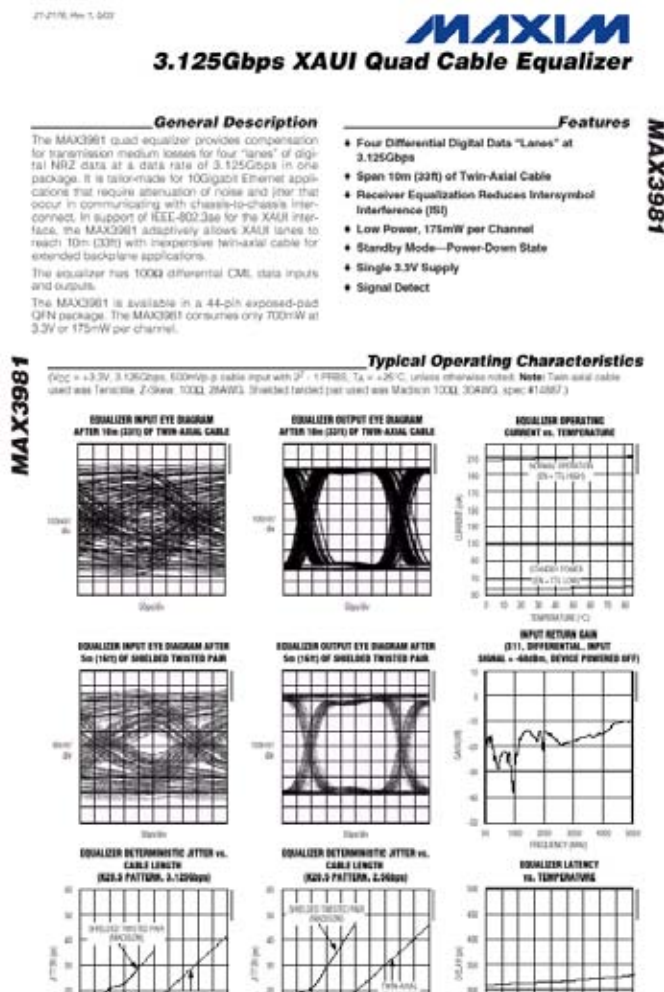


FIGURE 3-37: Sample Fact Sheet for an External 3G Equalizer

Cables can also be equalized. The most common cable equalization technique is to add a passive equalizing circuit in the cable assembly, usually in the connector.

Some higher-end cables obtain equalized-type characteristics through novel cable construction techniques involving silver plated solid copper cables (Figure 3-38).

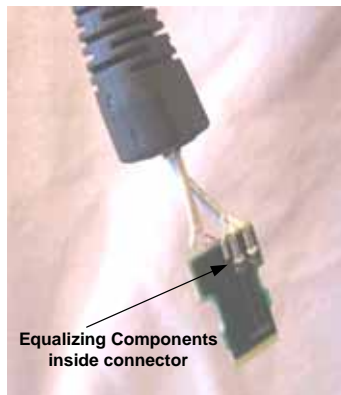


FIGURE 3-38: Equalized Cable Internal Components

Optical

The design solution will likely be optical if cables go much further than the adjacent chassis. With optical, there are a wide variety of optical choices to pass signals upstairs, across the building, around the block, or across town.

A basic optical system consists of a transmitter or source, the fiber, and a receiver.

Fiber optic systems use light instead of electricity to transport information. The basic systems consist of a transmitter or source, the fiber, and a receiver that converts the light pulse back into an electrical signal. The source is usually an injection laser diode (ILD) or a light emitting diode (LED) as shown in Figure 3-39.

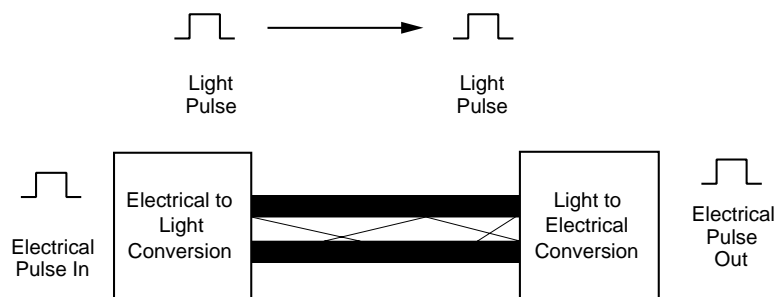


FIGURE 3-39: Basic Optic Transport System

Fiber allows transport of light pulses because of the principle of total internal reflection. This principle states that when the angle of incidence exceeds a critical value, light cannot get out of the glass. Instead, it bounces back in. In simple terms, fiber is like a long flexible paper towel-sized tube lined with a mirror and a flashlight. When shining a flashlight down the tube, even if the tube is bent around a corner, the light will continue to the end.

Total Internal Reflection: When the angle of incidence exceeds a critical value, light cannot get out of the glass. Instead, the light bounces back in.

There are two types of fiber—single-mode and multi-mode (Figure 3-40 and Figure 3-41). Single-mode is more expensive and allows for longer runs. Multi-mode is cheaper and can only be used for shorter distances. Basic optical connectors are shown in Figure 3-42.



FIGURE 3-40: SMF Single Path

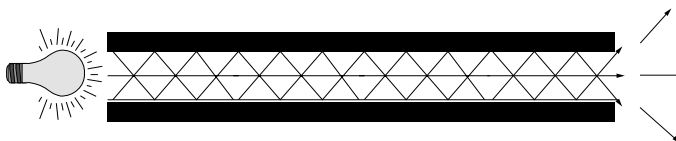


FIGURE 3-41: MMF Multiple Paths



FIGURE 3-42: Basic Optical Connectors

Bit Error Rate

The bit error rate (BER) is a concern for gigabit links designers, especially when moving from a parallel to serial backplane system. No link has a BER of zero because there is always some potential for errors. In many lower rate systems, the likelihood of errors is due to cosmic ray interference. And the likelihood of that is so small that it is essentially zero. So why are serial links different?

There are three reasons:

1. Cosmic rays can cause errors especially if they happen to hit during a transition. The faster the signal, the more transitions and the more likely a cosmic ray will occur during a transition.
2. For any given BER, the faster the signal, the more likelihood of an error.
3. High-speed clock data recovery is not an exact science. Jitter, ISI, and a host of other real world interferences can cause a bad data decision that results in an error. For example, PLLs, are constantly trying to adjust to the changing incoming signal. And as oscillators drift with temperature, errors can occur.

Realities of Testing

While the above reasons have a real effect, careful analysis of parallel backplanes, source synchronous links, or any communication channel could find similar faults. But for the most part, these are routinely assumed to be close to zero and ignored.

Why shouldn't we have the same concerns in Gigabit SERDES? Because their original environment is the communication industry with long and short haul optic transports. This is an industry that has always worried, tested, designed, and specified BERs. And this has had the most impact on Gigabit SERDES BER.

Some of the standards such as XAUI and some of the SONET variations specify a maximum BER. Unfortunately, testing for BER is difficult, boring, and time consuming. And it gets exponentially more difficult to get a unit of improvement. BERs are normally expressed in 10-x notation, so to move from 10-8 to a 10-9 takes 10-x the time. Testing becomes impractical at some point. Hence, most manufacturers test to the tightest BER in a published standard and no further.

CRC

But a designer must still design a system that is robust. To do so, he must first examine the system requirements to see if he can use the same commonly-used methods that contributed to the problem.

One method is error detection data retransmission. The incoming data is examined for errors. If any are discovered, a message is sent to the sender to retransmit. The preferred method for error detection is CRC. This is so common that many SERDES include CRC generation and checking hardware directly within the SERDES. Often, the retransmission request is built into an upper level protocol. This is the best solution if the protocol used supports CRCs and retransmission, or if the data requirements are such that they can be implemented.

If this is not possible, there are other options. The designer could simply build and test the system and see if it works. The published BER for the selected SERDES is a specification of how far it was tested, so the designer has some room to maneuver. It is possible that he can build a system far better than the published number. Besides being a specified testing stop point, the testing was probably done at the extremes (input jitter very near the maximum, etc.). If he designs the system to provide a better input stream, he will get better results.

Data offers another option to consider. Most data streams have a pattern and they are much more predictable than the pseudo-random bit streams used for BER testing. This can be good or bad depending on how the transmission path and equalizers react to the stream. This must be tested and adjusted.

So it is not completely far-fetched to build a system and see if it will work. However, if doing this presents a management concern, forward error correction (FEC) can alleviate concerns.

FEC Used in Some Applications

Since the designer knows that errors are going to occur, he can prepare to recover from those error by providing extra data bits.

FEC: Forward Error Correction—Extra bits are added to data to help recover from an error.

Let's examine how FEC works. Consider a block of data to be transmitted $N \times R$ bytes long and divide it into a matrix N bytes by R rows. Now add one extra byte to each row and one extra row to the matrix. These are the extra slots.

Additional information about the data block will be put in these slots. In this example, the extra information is parity bits. Each bit of the extra byte on each row represents the parity of that specific bit for each byte on the row. That is, $P[1][0]$ is the parity of $D[1.1][0]$ $D[1.2][0]$ $D[1.3][0]$... $D[1.N][0]$. Then for the extra row, the parity of the bits directly above are taken. That is, $P[R+1.0][0]$ is the parity of $D[0.0][0]$, $D[1.0][0]$ $D[2.0][0]$, $D[N.0][0]$. A diagram of this matrix is shown in Figure 3-43.

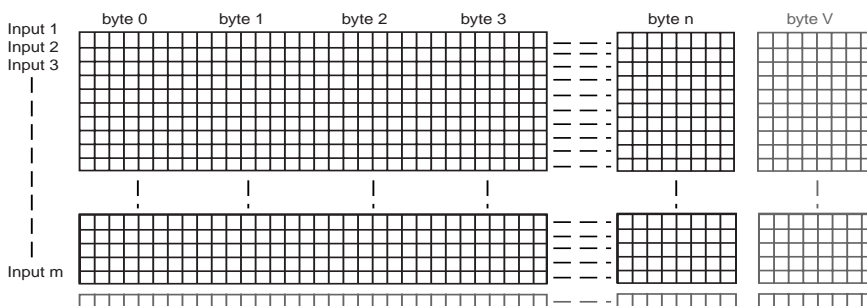


FIGURE 3-43: FEC Diagram

The data and extra bits are transmitted over the link. On the other side, the matrix is examined for parity errors. If any one bit of data is the wrong value, it will be flagged and identified by row and column. This bit could then be corrected by a simple inversion. Depending on where errors occur, multiple errors could either be corrected or they could cause confusion and prevent the correction of other errors.

This method is known as a simple parity matrix and was the first type of FEC. It is the basic building block for most FEC methods. While this example is straightforward, it does have limitations. Some FEC methods have been developed for harsh environments or dirty channels like Viterbi, Reed-Soloman, or Turbo Product codes. All have powerful correction, but that correction comes at a cost:

- **They do not go very fast.** Gigabit SERDES are faster than most of these methods can handle in their normal construction.
- **They are too big.** The encoders and decoders may consist of ten times as much logic as the MGT and/or the rest of the design.
- **The coding overhead is too great.** The coding overhead is the added bits. Often the coding overhead can completely eliminate the feasibility of the FEC method.

SERDES Technology Facilitates I/O Design

I/O design is facilitated by tapping the functions available in SERDES technology. SERDES functions such as RX align, clock manager, transmit/receive FIFO, and line encoder/decoder are used extensively to improve speed and accuracy. As SERDES plays a more important role in future I/O designs, its functions will continue to provide tools for more efficient I/O devices.