High Speed Serdes Devices and Applications

David R. Stauffer • Jeanne Trinko Mechler
Michael Sorna • Kent Dramstad
Clarence R. Ogilvie • Amanullah Mohammad
James Rockrohr

High Speed Serdes Devices and Applications

Springer

David R. Stauffer
IBM Corporation
Essex Junction, VT
USA

Kent Dramstad
IBM Corporation
Essex Junction, VT
USA

Amanullah Mohammad
IBM Corporation
Research Triangle Park, NC
USA

Michael A. Sorna
IBM Microelectronics
Hopewell Junction, NY
USA

Jeanne T. Mechler
IBM Corporation
Essex Junction, VT
USA

Clarence R. Ogilvie
IBM Corporation
Essex Junction, VT
USA

James D. Rockrohr
IBM Microelectronics
Hopewell Junction, NY
USA

ISBN 978-0-387-79833-2 e-ISBN 978-0-387-79834-9

Library of Congress Control Number: 2008925643

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

Preface

The simplest method of transferring data through the inputs or outputs of a silicon chip is to directly connect each bit of the datapath from one chip to the next chip. Once upon a time this was an acceptable approach. However, one aspect (and perhaps the only aspect) of chip design which has not changed during the career of the authors is Moore's Law, which has dictated substantial increases in the number of circuits that can be manufactured on a chip. The pin densities of chip packaging technologies have not increased at the same pace as has silicon density, and this has led to a prevalence of High Speed Serdes (HSS) devices as an inherent part of almost any chip design.

HSS devices are the dominant form of input/output for many (if not most) high-integration chips, moving serial data between chips at speeds up to 10 Gbps and beyond. Chip designers with a background in digital logic design tend to view HSS devices as simply complex digital input/output cells. This view ignores the complexity associated with serially moving billions of bits of data per second. At these data rates, the assumptions associated with digital signals break down and analog factors demand consideration. The chip designer who oversimplifies the problem does so at his or her own peril.

Despite this, many chip designers who undertake using HSS cores in their design do not have a sufficient background to make informed decisions on the use of HSS features in their application, and to appreciate the potential pitfalls that result from ignoring the analog nature of the application. Databooks describe the detailed features of specific HSS devices, but usually assume that the reader already understands the fundamentals. This is the equivalent of providing detailed descriptions of the trees, but leaving the reader struggling to get an overview of the forest.

This text is intended to bridge this gap, and provide the reader with a broad understanding of HSS device usage. Topics typically taught in a variety of courses using multiple texts are consolidated in this text to provide sufficient background for the chip designer that is using HSS devices on his or her chip. This text may be viewed as consisting of four sections as outlined below.

The first three chapters relate to the features, functions, and design of HSS devices. Chapter 1 introduces the reader to the basic concepts and the resulting features and functions typical of HSS devices. Chapter 2 builds upon these concepts by describing an example of an HSS core, thereby giving the reader a concrete implementation to use as a framework for topics throughout the remainder of the text. Although loosely based on the HSS designs offered in IBM ASIC products, this HSS EX10 is a simplified tutorial example and shares many features/functions with product offerings from other vendors. Finally, Chap. 3 introduces interested readers to the architecture and design of HSS cores using the HSS EX10 as an example.

The next two chapters describe the features and functions of protocol logic used to implement various network protocol interface standards. Chapter 4

introduces concepts related to interface standards, as well as design architectures for various protocol logic functions. Chapter 5 provides an overview of various protocol standards in which HSS cores are used.

The next four chapters cover specialized topics related to HSS cores. Chapter 6 describes clock architectures for the reference clock network which supplies clocks to the HSS core, as well as floorplanning and signal integrity analysis of these networks. Chapter 7 covers various topics related to testing HSS cores and diagnostics using HSS cores. Chapter 8 covers basic concepts regarding signal integrity, and signal integrity analysis methods. Chapter 9 covers power dissipation concepts and how these relate to HSS cores.

Finally, any HSS core is not complete without a set of design kit models to facilitate integration within the chip design. Chapter 10 discusses various topics regarding the design kit models that require special consideration when applied to HSS cores.

Acknowledgments

The authors wish to thank the following IBM colleagues without whose contributions and reviews this text would not be possible: William Clark, Nanju Na, Stephen Kessler, Ed Pillai, M. Chandrika, Peter Jenkins, Douglas Massey, Suzanne Granato, Della Budell, and Jack Smith.

In addition, the authors would like to thank Thucydides Xanthopoulos of Cavium Networks for his detailed and insightful review of this text, and Andrea Kosich for making it possible to utilize material from Optical Internetworking Forum Interoperability Agreements.

Table of Contents

Chapter 1
Serdes Concepts

This chapter describes basic methods of transferring data from one chip to another chip, either on the same circuit board or across a cable or backplane to another circuit board. After reading this chapter, the reader should have a basic understanding of the rationale for using high-speed serializer/deserializer (Serdes) devices, and the inherent problems introduced by the high-speed operation of such devices.

1.1 The Parallel Data Bus

The simplest method of transferring data through the inputs or outputs of a silicon chip is to directly connect the datapath from one chip to the next chip (see Fig. 1.1). Since data often consists of more than one bit of information, the datapath is more than one-bit wide. In the figure, an *n*-bit datapath inside *Chip #1* is driven through chip outputs, across an *n* bit interconnect, through inputs of *Chip #2*, to an *n*-bit datapath inside the receiving chip. Synchronous data is transferred between the two chips since both chips are clocked by the same clock source.

There are two inherent problems of the parallel data bus shown in Fig. 1.1. The first problem is that *n* input/output (I/O) pins are required on each chip to transfer the data. At one point in history this was acceptable. However, Moore's Law has driven substantial increases in the number of circuits that can be manufactured on a chip compared to a few decades ago. The pin densities of chip packaging technologies have not increased at the same pace as silicon density. Therefore, I/O pins are substantially more expensive than silicon circuits, and dedicating *n* I/O pins for the above data bus is not acceptable for most chip applications.

The second inherent problem involves meeting timing requirements. The data is launched synchronously by *Chip #1* and is captured synchronously in *Chip #2* using the same clock. The data at the inputs of *Chip #2* must meet setup and hold times relative to the clock input of the chip. These setup and hold times must be calculated with sufficient margin to allow for differences in delay of the clock distribution path to the two chips, and through the chips to the launch and capture flip-flops. Delays may vary based on chip process, voltage, and temperature (PVT) conditions, and margin must be added to account for worst case variations. For higher clock frequencies, it may be necessary to use phase-locked loops (PLLs) in the chips to adjust the clock phase in order to compensate for the clock distribution delay within the chip and adapt to changing process, voltage, and temperature conditions. If the clock frequency is high enough, it will not be possible to build a system that will reliably transfer the data across this data bus.

D. R. Stauffer et al., *High Speed Serdes Devices and Applications,*

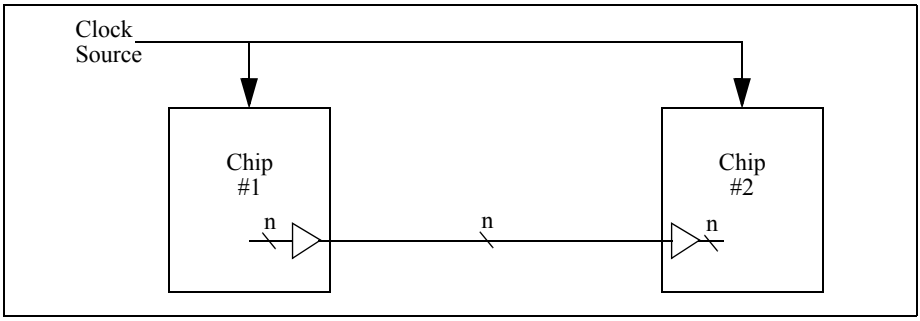


Fig. 1.1 Parallel data bus between two chips

1.2 Source Synchronous Interfaces

The two problems with the parallel data bus in Sect. 1.1 can be eliminated with the modifications to the system which are discussed in this section. These approaches are extensions of the parallel data bus. The parallel data bus and all of the extensions described in this section are considered to be *source synchronous interface* architectures. Such architectures include any interface where a clock input exists that can be used to capture the received data. This may be either a reference clock used by both the transmitting and receiving chip or the transmitting chip drives a clock to the receiving chip. In either case, clock recovery circuits are *not* required for source synchronous interfaces.

1.2.1 Reducing the Number of I/O Pins

The first issue to be addressed is reducing the number of I/O pins required to transfer the data between the chips. This is accomplished by multiplexing the n bits of data at the output of *Chip #1* onto k bits of interconnect ($k < n$), and then demultiplexing the k bits of interconnect at the input of *Chip #2* onto an n bit internal datapath. This is shown in Fig. 1.2. The resulting system only requires k I/O pins on each chip rather than the n pins previously required.

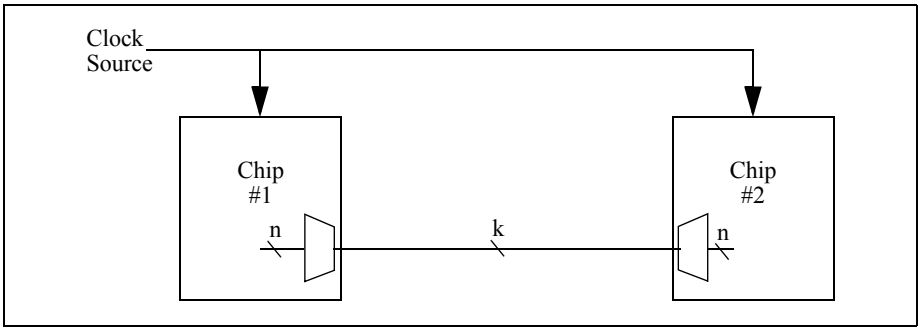


Fig. 1.2 Serializing the data to reduce pin counts

Of course, while the pin count requirements have been reduced by the ratio of $k : n$, the required frequency of the reference clock has increased by the inverse of this ratio. System designers generally do not like to distribute high-speed reference clocks within the system due to noise, electromagnetic interference (EMI), and power dissipation concerns. Often, a lower frequency clock is distributed, and PLLs in the chips are used to multiply this reference clock to a usable frequency. Variability of the phase of the resulting clock, along with the higher frequency of data transfer, tends to exacerbate the timing issues of the parallel data bus approach.

1.2.2 Clock Forwarding

In Fig. 1.3, a high-speed clock has been added to the datapath between the two chips. This clock source is assumed to supply a clock frequency somewhat lower than the frequency required to clock the data flip-flops on the chip interconnect. PLLs are used in each chip to generate clocks at a multiple of this frequency. The resulting clocks are used to launch and capture data in the respective chips. The output clock of the PLL in *Chip #1*, which is used to launch the data from this chip, is also an output of this chip. This clock is used by *Chip #2* to capture the data. This approach is called *clock forwarding*.

The advantage of this approach is that the high-speed clock used to launch the data at *Chip #1* is available to *Chip #2* as a reference to capture the data. Any variations in delays through clock distribution network driving the two chips does not need to be taken into account in timing analysis. Only delay variations between the clock path and the data bits are relevant. Variations between these paths due to process, voltage, and temperature track each other to some extent. The result is that timing analysis of the interface requires less margin and setup and hold times are therefore easier to meet.

So far we have not made any distinction or recommendations regarding the frequency of the high-speed clock relative to the bit rate of the interface. In general, the high-speed clock shown in the figure could be *single data rate* (SDR) or *double data rate* (DDR) (Fig. 1.4). The receiving chip captures data on every rising (or every falling) edge of an SDR clock; while the receiving chip captures data on every edge (both rising and falling edges) of a DDR clock.

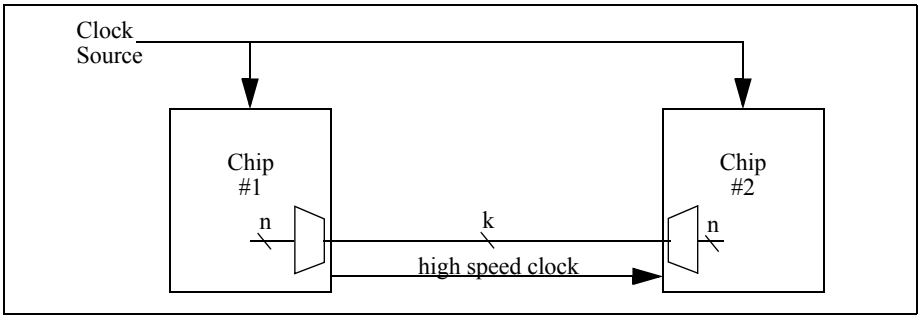


Fig. 1.3 High-speed clock forwarded with the data

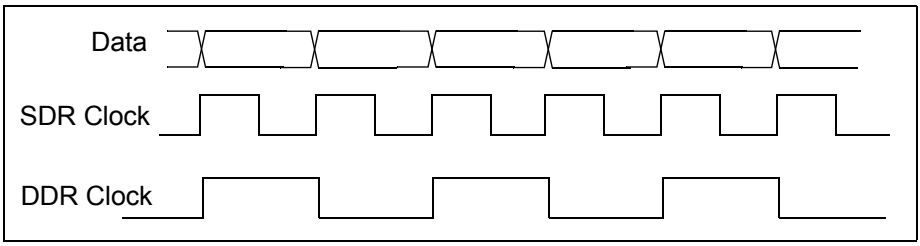


Fig. 1.4 Single data rate and double data rate clocks

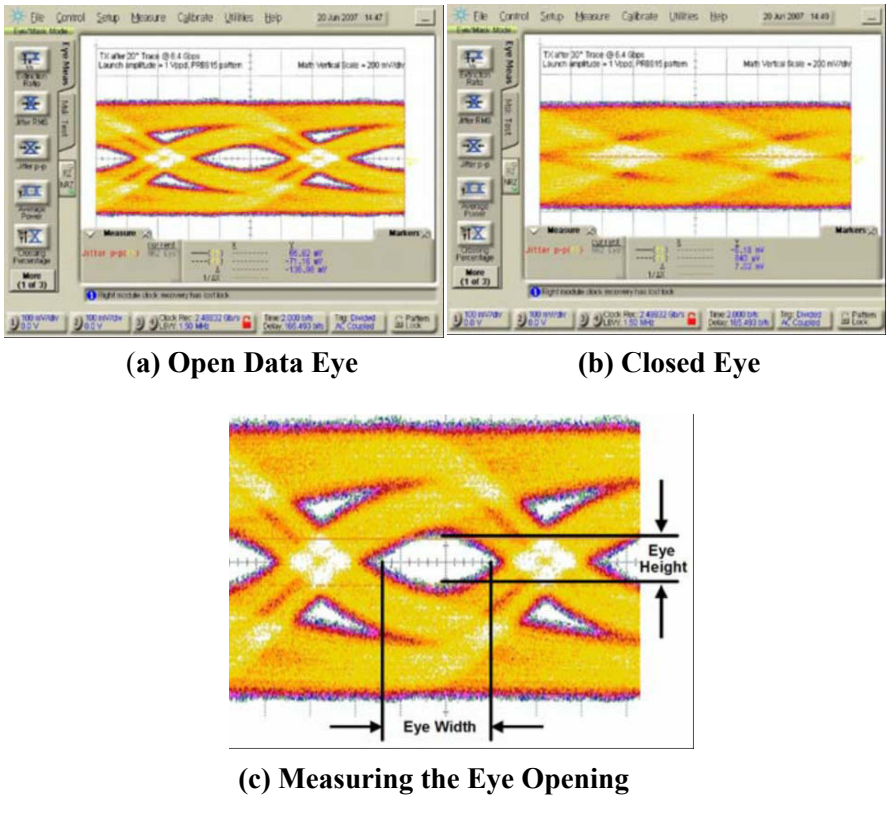
The advantage of DDR clocks over SDR clocks lies in the bandwidth requirements for the corresponding I/O drivers and receivers. An I/O cell being used at a bit rate of *b* Mbits per second requires a bandwidth sufficient to transmit a 101010... data pattern. This corresponds to a frequency spectrum with an upper fundamental frequency limit of *b*/2 MHz. The corresponding frequency of an SDR clock is *b* MHz, twice the spectral limit of the data. However, the frequency of a DDR clock for the same interface is only *b*/2 MHz, consistent with the frequency spectrum of the data. Therefore, the same I/O drivers and receivers can be used to drive and receive both the data and the DDR clock.

Regardless of whether the high-speed clock is a SDR or a DDR clock, the receiving chip uses this clock to directly capture the data. This chip also uses the reference clock to generate an internal system clock at the same frequency. These clocks are mesochronous. While the frequency is the same (given that they share a common frequency reference), the phase relationship between the clocks is unknown and may vary due to PVT variations. Therefore, the receiving chip usually retimes the received data from the interface clock domain to the clock domain of the internal chip clock. FIFOs are used to perform this retiming function. It is desirable to minimize the number of flip-flops being clocked by the interface clock in order to minimize delay in the clock distribution network; otherwise timing issues will be exacerbated.

1.2.3 Higher Speed Source Synchronous Interfaces

The window of time during which data bits can be assumed to be valid is called the *eye*. This name originates from the shape of the waveform when the data signal is monitored on an oscilloscope that is continuously triggered. An example of a serial data eye is shown in Fig. 1.5a. *Eye closure* results from process, voltage, and temperature effects, as well as differences between signal rise and fall times, slew rates, etc. The more the eye is closed, the more difficult it is to find a point at which the signal can be reliability sampled to receive the data. The serial data eye shown in Fig. 1.5b is completely closed.

The largest possible *eye opening* is desirable. The width and height of an open eye can be measured as shown in Fig. 1.5c. The expected bit error rate (BER) of the link directly correlates to the amount of eye opening (both width and height). This section briefly describes some approaches to minimize eye closure of the data signal. Eye waveforms are discussed further in later chapters.



(a) Open Data Eye **(b) Closed Eye**

(c) Measuring the Eye Opening

Fig. 1.5 Example of a serial data eye

1.2.3.1 Differential Signals

Unequal signal rise and fall times of nondifferential signals contribute to eye closure. Signals switching on the chip also create current variations on the power distribution grid of the chip, which in turn cause variations in voltage drop (noise) that can cause variation in delays of surrounding circuits. One method of reducing the effects of these phenomenon on the eye width is to drive differential signals between chips.

Differential signals represent the data bit using two electrical signals (*true* and *complement* signals). A logic "0" is represented by the *true* signal driven to its lower voltage limit, and the *complement* signal driven to its upper voltage limit; a logic "1" is represented by the *true* signal driven to its upper voltage limit, and the *complement* signal driven to its lower voltage limit. A differential receiver device interprets the logic bit value based on the difference between the two signals, and not based on the level of either signal individually.

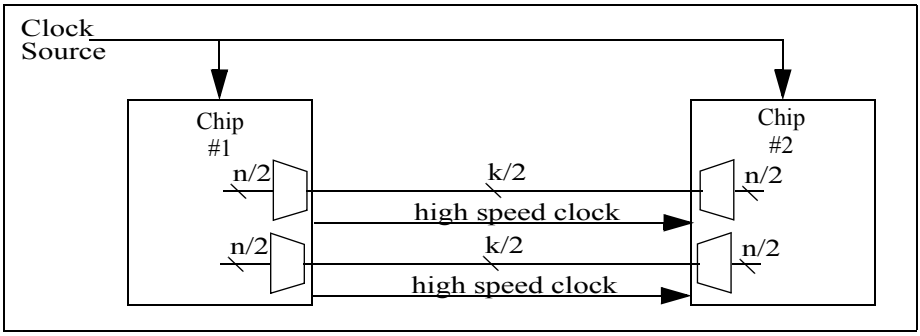


Fig. 1.6 Multiple sets of data with separate high-speed clocks

Differential driver circuits tend to have linear current draw and generate less noise on the power supply than equivalent single-ended drivers. Most noise sources induce voltage variation equally on both the true and complement signals; such *common mode noise* is ignored by the receiver. Also, since one leg of the differential signal is rising while the other is falling, or vice versa, unequal rise and fall time effects cancel.

The drawback of differential signals is that two chip pins are required for each data bit. However, this is offset by the higher speeds possible with differential signals that are not possible with single-ended signals.

1.2.3.2 Multiple Interface Clocks

The interface clock in Fig. 1.3 is the same clock as is used to launch the data, and in general is driven from a point in the clock distribution network as close to the actual flip-flops that launch the data as possible. Phase variation is introduced by any circuits which are not common to both the data path and clock path. Silicon process variables do vary from circuit to circuit on the same chip, the power distribution network may have unequal voltage drops to different circuits which may vary based on switching currents, and the temperature may vary from point to point on the chip. Tolerances and limits for all of these parameters must be taken into account when calculating delays, setup times, and hold times necessary for correct capture of the received data. At higher bit/baud rates, these parameters may significantly reduce the eye opening, and become the dominant mechanism for limiting the speed of the interface.

To maximize the eye width, the path through the clock tree to each of the data flip-flops and to the clock output should share as many circuits as possible, and the output driver for the clock should be similar to the output drivers for the data. Ideally, the same clock buffer should drive the clock to the output driver and should drive the clock input to all of the data flip-flops. The larger the number of bits in the data bus, the more difficult this becomes to implement. I/O drivers must be physically distributed based on the groundrules for connections to package pins. The greater the distance between circuits, the more process, voltage, and temperature variation, and the more circuits in the clock distribution network which cannot be shared due to lack of proximity.

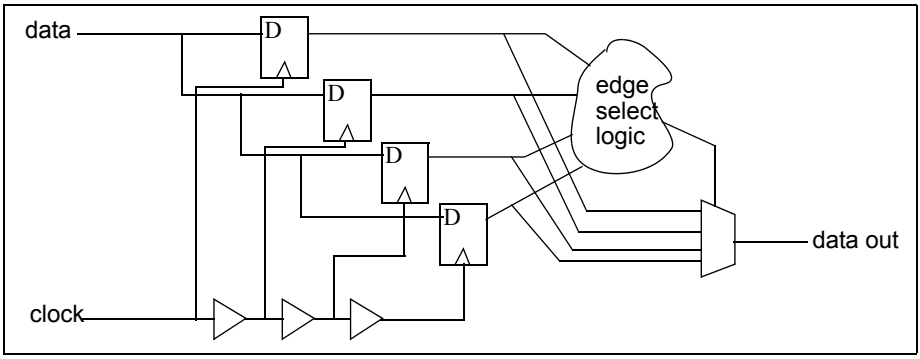


Fig. 1.7 Adapting the sampling clock phase in the receiver

One technique used to improve eye width is to limit the number of data bits associated with a given interface clock line. Wider data busses are built by using multiple interface clocks, each clock associated with a subset of the data bits. An example of this is shown in Fig. 1.6, where the *k* bit interconnect has been subdivided into two groups, each with its own high-speed interface clock. Note that the receiving chip must capture each group of data bits in separate clock domains, and needs to retime this data to the common clock domain internal to the chip.

1.2.3.3 Sample Edge Adaptation

Another technique used to permit higher speed operation of source synchronous interfaces is to process the data signal at the receiver and adapt the sampling phase of the clock on a per-bit basis. This is done by connecting the received interface clock signal to the input of a multitap delay line, and capturing the data signal in multiple flip-flops clocked by different clock phases. Logic can then be used to determine the clock phases between which data transitions are occurring, and select the optimal clock phase to be used to capture the data. This scheme is shown in Fig. 1.7.

Schemes, such as shown in Fig. 1.7, may require a training pattern either upon initialization of the interface or at regular intervals. If a training pattern is used, phase selections remain static between training periods. More complex implementations adjust dynamically based on the received data or based on training patterns embedded in the data stream. Alternative architectures which apply the data to the delay line are also possible. Note, however, that an inherent characteristic of most of these schemes is that the phase adjustment is less than plus/minus one bit time, and there must be a sufficient eye opening such that an optimal sampling phase exists.

Given the advanced schemes discussed above, data rates for source synchronous interfaces can be extended to several Gigabits per second (Gbps) per interconnect bit. However, PVT variations make further increases in interface speeds prohibitively complex. Beyond these speeds, High-Speed Serdes devices that extract the clock from edge transitions in the data stream become the preferred solution.

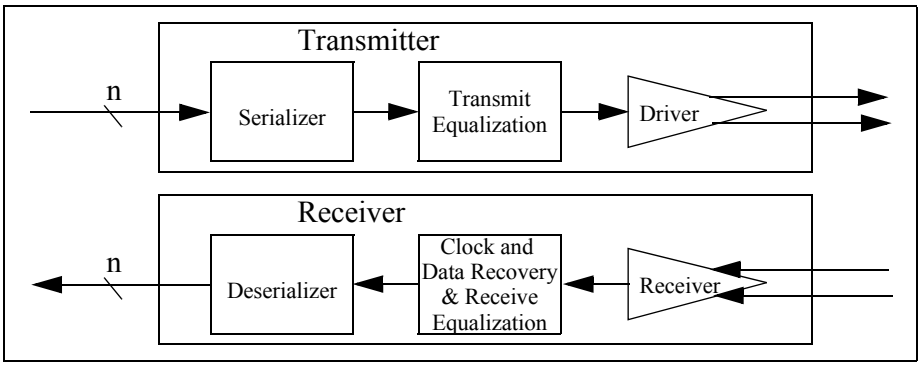


Fig. 1.8 Basic block diagram of typical high-speed serdes

1.3 High-Speed Serdes

High-Speed Serializer/Deserializer (HSS) devices are the dominant implementation of I/O interfaces at speeds of 2.5 Gbps and higher. Such devices are differentiated from source–synchronous interfaces in that the receiver device contains a clock and data recovery (CDR) circuit which dynamically determines the optimal sampling point of the data signal based upon the transition edges of the signal. In other words, clock information is extracted directly from the data rather than relying on a separate clock.

Figure 1.8 illustrates the basic block diagram of the transmit and receive channels of an HSS device. The transmitter serializes parallel data, equalizes it for reasons that will be explained shortly, and then drives the serial data onto a differential signal pair of interconnect wires. Feed forward equalizers (FFE) are commonly used in High-Speed Serdes devices, as discussed in Sect. 1.3.2. The receiver consists of a differential receiver, a CDR circuit which may also integrate an equalizer, and deserializes the data based upon the sample point established by the CDR. Peaking amplifiers and/or decision feedback equalizers (DFE) are commonly used for equalization in High-Speed Serdes receiver devices.

Note that Serdes cores are often designed to group multiple transmit and/or receive channels into a single device. The individual channels generally operate independently. Grouping channels allow some circuits to be shared across channels (for example the PLL noted below), and therefore the resulting block is more efficient in terms of chip area, cost, and power.

Serdes cores which contain only transmit or only receive channels are called *simplex* cores; Serdes cores which contain both transmit and receive channels are called *full duplex* cores. Note that the terminology “full duplex” does not imply that the electrical interface is bidirectional. Any given electrical interconnect channel has a fixed direction of data transmission. If a protocol application requires “full duplex” communication, then independent transmit and receive channels with independent interconnections are used to implement the interface. Rationale for using simplex vs. full duplex cores may include

(1) chip floorplan to minimize wiring crossings in the package design or circuit board design; (2) signal integrity concerns due to near-end crosstalk from transmit signals onto receive signals; (3) or applications where the number of transmit and receive channels is not equal.

The remainder of this section generically describes various circuits mentioned above in more detail, as well as providing generic descriptions of other circuits and functions commonly found in High-Speed Serdes cores.

1.3.1 Serializer/Deserializer Blocks

Conceptually, the input to the serializer transmit stage is an *n*-bit datapath which is serialized to a one-bit serial data signal for application to the FFE and Driver stages. Generally the value of *n* is a multiple of 8 or 10, and may be programmable on some implementations. Values of *n* which are multiples of 8 are useful for sending unencoded and/or scrambled data bytes; values of *n* which are multiples of 10 are useful for protocols which use 8B/10B coding, as discussed further in Sect. 4.2.2.1. (The 8B/10B encoder is generally implemented by logic outside the Serdes core.)

For simplicity, the block diagram in Fig. 1.8 illustrates the serializer feeding one-bit data into the transmit equalization block. Actual implementations may vary, and this datapath may be one or more bits wide. A wider datapath through the equalizer block results in a more complex design, but requires a lower operating frequency. Some implementations may initially multiplex the *n*-bit input to an *m*-bit datapath ($m < n$) prior to the equalizer, and perform the remainder of the serialization at the driver stage.

The serializer stage latches data on the *n*-bit input at the frequency of *baud rate*/*n*. The high-speed clock in the Serdes is divided down to generate a sample clock for the parallel data. Because the phase of this clock is determined by the internal state of the serializer, the Serdes channel generally provides this clock as an output for use by logic driving data to the transmit channel.

Conceptually, the deserializer receive block performs the inverse function of the serializer block. Serial data is deserialized onto an *n*-bit databus of similar width to the serializer. A sample clock is generated by dividing down the internal high-speed clock, and this clock is supplied as an output for use by logic latching the parallel data. In a similar manner to the serializer, actual implementations may perform partial deserialization in a prior stage.

Many Serdes receivers also include a feature to assist with data alignment of the output. Most applications organize data into bytes or words (groups of bytes). For 8B/10B encoded applications, data is organized into 10-bit encoded symbols. The initialization of the clock divider in the deserializer is arbitrary, and the data received on the parallel data bus will have an arbitrary alignment that is unlikely to match the byte or symbol boundaries of the protocol. This can be corrected by downstream logic to steer data onto the appropriate byte, symbol, or word boundary. Alternatively, many Serdes receivers provide an input which forces the deserializer to “slip” one bit. Downstream logic detects

that data is not aligned to the appropriate boundary, and repeatedly pulses the deserializer control until the data "slips" to the desired alignment.

1.3.2 Equalizers

The interconnect between the transmitter and receiver device (known as the *channel*) acts as a filter at typical baud rates, and distorts the serial data signal to varying extents. Figure 1.9 illustrates this distortion: The input waveform is a clean digital signal, but the output waveform is significantly distorted. The illustrated frequency response function for the channel is characteristic of a low-pass filter. Signal distortion occurs because the signal baud rate is above the cut-off frequency for this filter.

Signal integrity concerns frequently dictate that the data signal be equalized at the transmitter and/or receiver in order to counter the effects of the channel and decode the signal properly. Many variations on filter architectures are possible, all of which accomplish this. Fig. 1.10 illustrates the addition of an equalizer at the transmitter with a transfer function that is roughly the inverse of the channel's frequency response. This equalizer distorts the signal at the transmitter output such that the resulting signal at the receiver input is a clean waveform.

Most Serdes transmitter implementations include a FFE. The block diagram for a three-tap FFE is shown in Fig. 1.11. The serial data signal is delayed by several flip-flops which implement the *taps* for the filter. Each tap is multiplied by a *tap weight* value (also called a *filter coefficient*), and the results are summed and driven to the serial data output. FFE operation is described further in Sect. 3.2.1.

The number of FFE taps on the filter, the spacing of these taps relative to the baud rate, and the granularity of these tap weight values vary based on implementation. The terminology *preemphasis* or *deemphasis* refer to the FFE architecture, and indicate whether the data signal amplitude is increased or decreased as compared to the nonemphasized value by the FFE tap. The terminology *precursor taps* and *postcursor taps* refer to whether the FFE filter taps operate on an advanced or delayed signal (respectively) relative to the $t = 0$ tap. *Baud-spaced* taps are defined as taps where the delay from one filter tap to the adjacent tap is one-bit time interval; fractional spacing of the taps is also possible.

The FFE tap weights are selected to generate a filter with the inverse transfer function of the channel transfer function. Various algorithms exist for determining optimal FFE coefficient values; some select filter coefficients to maximize signal amplitude at the receiver, while others optimize eye width (i.e., minimizing jitter). More complex algorithms may search for an optimal trade-off between amplitude and jitter in order to optimize a more complex parameter (such as projected BER). FFE tap weights are determined for many applications by design and coded as fixed values within system software, however, there are some applications where FFE tap

weights are adjusted dynamically by the protocol based on signal characteristics at the receiver.

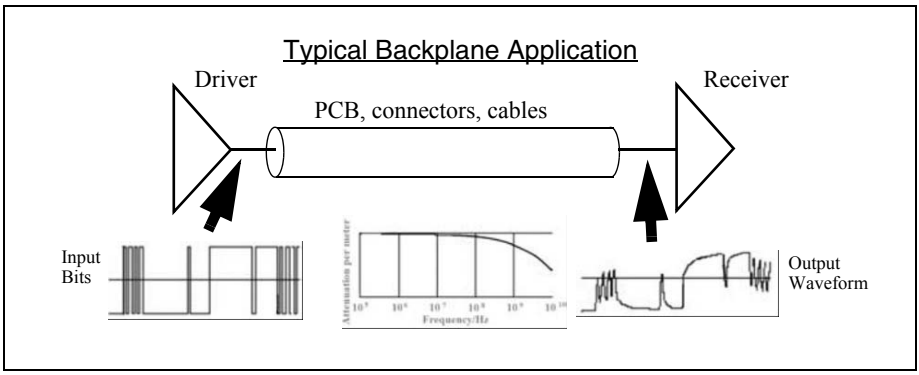


Fig. 1.9 Signal distortion for a typical channel application

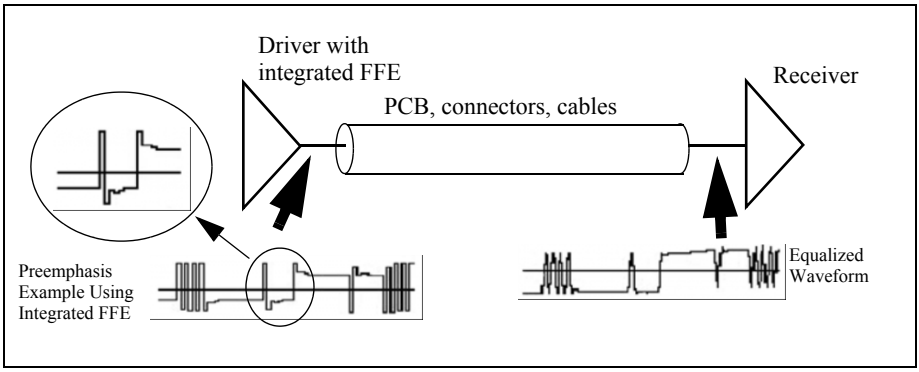


Fig. 1.10 Typical channel application with equalization at the transmitter

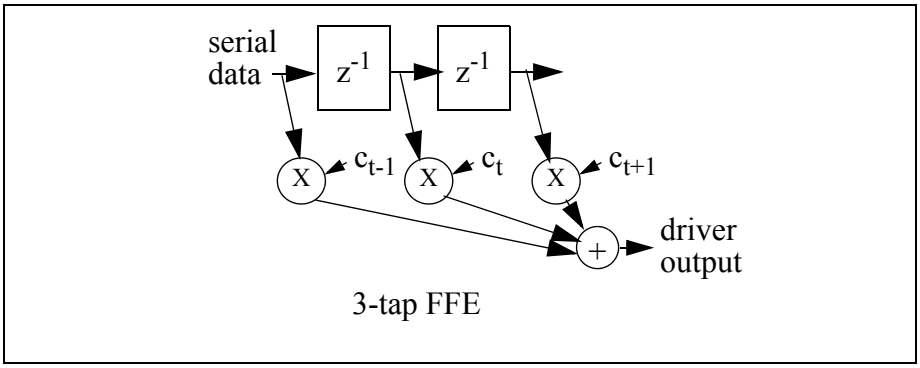


Fig. 1.11 Three-tap feed forward equalizer operation

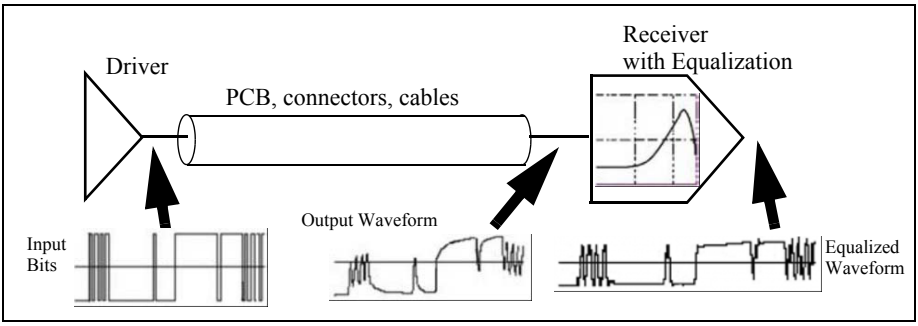


Fig. 1.12 Typical channel application with equalization at the receiver

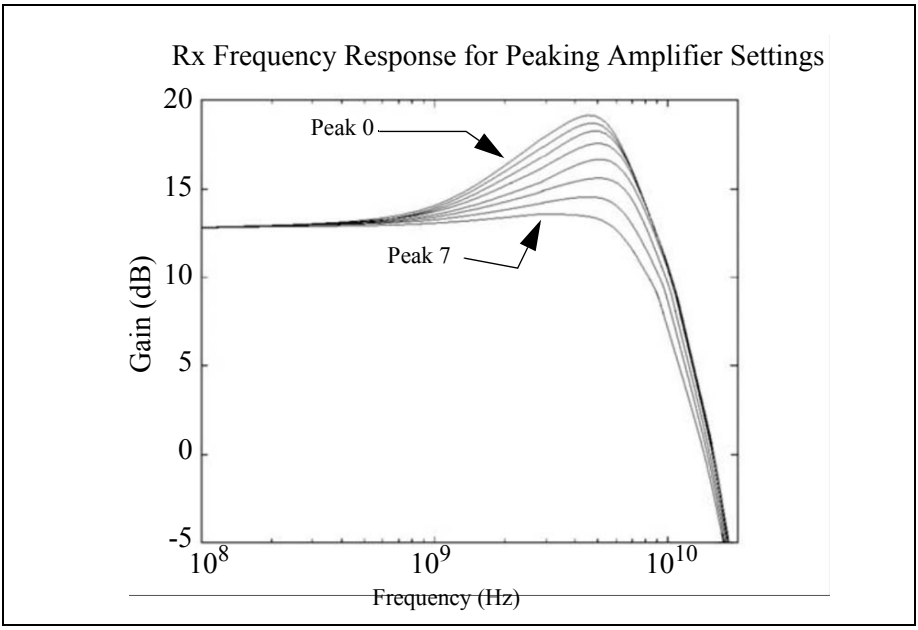


Fig. 1.13 Receiver frequency response for peaking amplifier settings

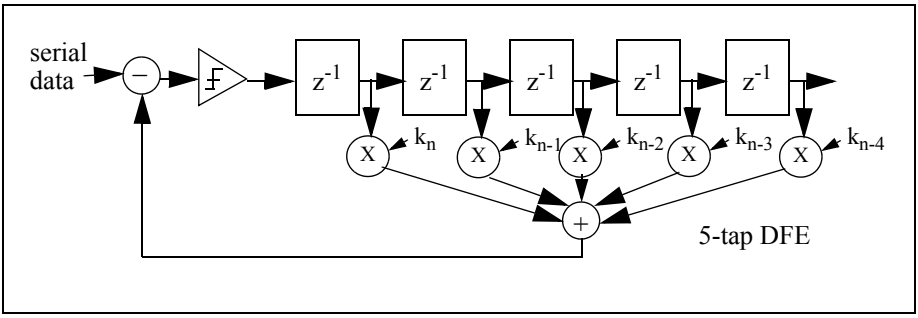


Fig. 1.14 Decision feedback equalizer architecture

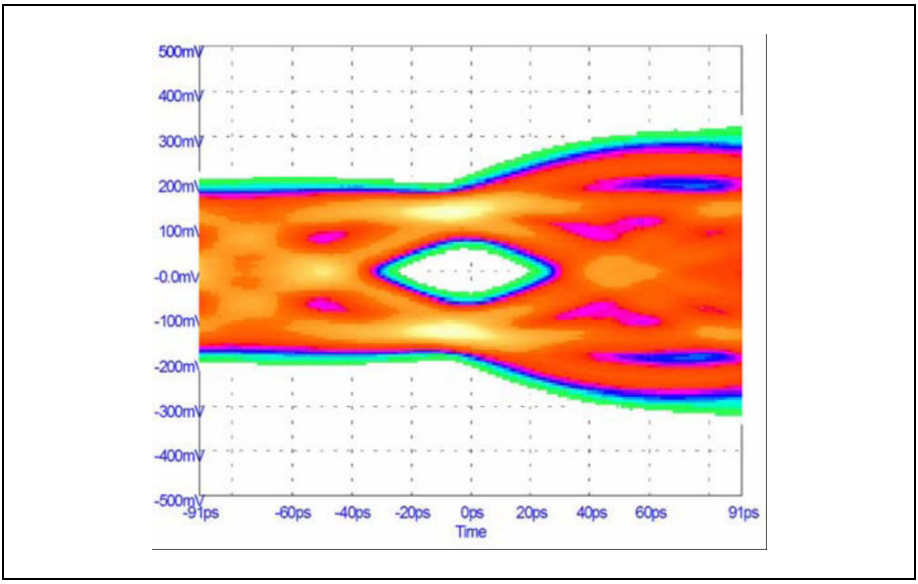


Fig. 1.15 Virtual eye after equalization

Equalization may also be performed at the receiver as illustrated in Fig. 1.12. Despite the signal distortion at the input of the receiver, this equalizer corrects for the distortion and produces a clean waveform. For lower speed or lower loss links, the most prevalent approach is to use some variant of a peaking amplifier. Peaking amplifier circuits amplify the higher frequency signal components more than the lower frequency components. If the peaking amount is matched to the high frequency loss (difference between high frequency and low frequency), then the channel is equalized and the eye is opened up. Some Serdes devices allow programmable peaking levels; the frequency response of such a peaking amplifier for various provisioned settings is shown in Fig. 1.13.

For higher baud rates, the transfer function of the channel can cause jitter exceeding the bit width of the data and significant loss of signal amplitude at higher frequencies. A DFE stage is often included in receivers for these baud rates in order to recover data despite the otherwise "closed" eye.

A conceptual block diagram of a DFE circuit is shown in Fig. 1.14. The serial data signal is applied to a slicer circuit which makes decisions as to whether the incoming signal is a "0" or a "1". The received serial data is then delayed by a number of flip-flops which implement the filter taps. Each tap is multiplied by a corresponding tap weight value, and the results are summed. This sum is then used to correct the amplitude of the incoming signal, affecting the decisions made by the slicer circuit. Slicer decisions are thus affected by feedback based on prior data received. Although not shown in Fig. 1.14, some DFE architectures use feedback to affect both the amplitude and the sample

time of the slicer circuit. Such architectures adjust the CDR sample time from bit to bit based on feedback regarding the last several bits received.

As was the case for FFE circuits, the number of DFE taps on the filter, the spacing of these taps relative to the baud rate, and the granularity of these tap weight values vary. DFE implementations usually also contain logic which trains the DFE and sets DFE tap weights to optimal values dynamically.

Using a DFE, the closed eye shown in Fig. 1.5b is cleaned up to produce the virtual eye shown in Fig. 1.15. Note that the eye in this illustration is produced by a DFE architecture which corrects the CDR sample time from bit to bit. The DFE correction is valid for only one instance in time (based on the history of the previous bits). As such, once the DFE makes a decision as to whether the bit is "0" or "1", the DFE then proceeds to make adjustments for the next bit time which are different for the various signal traces of the composite waveform. For this reason, the signal eye shown in the figure is open for the bit of interest, but does not appear open for adjacent bits.

Many variations on equalizer architectures exist. As the baud rate increases, equalizer architectures become increasing complex. In some cases, protocol standards specify a base level of required equalizer functionality.

1.3.3 Clock and Data Recovery (CDR)

Conceptually, CDR circuits monitor transitions of the data signal and select an optimal sampling phase for the data at the mid-point between edges. Since the timing of data transitions includes a jitter component, the CDR must perform some averaging to provide stability of this sampling point from one bit to the next. Intersymbol interference (ISI) and other components of deterministic jitter (DJ) are dependent on the spectral content of the data signal, and this frequency spectrum does change based on the data content. Shifts in this frequency spectrum sustained for hundreds of bits or more cause the CDR to adjust the optimal sampling phase dynamically.

CDR architecture is discussed further in Sect. 3.3.1. Features of the CDR may be of some significance to the Serdes user are discussed below.

1.3.3.1 Maximum Run Length

A significant parameter for the Serdes which is primarily the result of the CDR design is the maximum number of consecutive "0" or "1" bits which can be received before the sampling point of the CDR risks incorrectly sampling the bits. An excessively long run of consecutive bits of the same value means that the CDR is not detecting any data transitions, and therefore cannot recover any clock information to ensure the data continues to be sampled in the center of the eye. A small drift in the sampling point relative to the baud rate of the data may cause the CDR to sample more "0" or more "1" bits than were actually transmitted. Also, the sampling point may require recentering when data transitions resume, and additional bits may be sampled incorrectly as this adjustment occurs. Some CDR implementations drive the receive data to a PLL and use the output of the PLL as the sample clock; clock outputs of the

receiver may change frequency or stop when such CDRs do not receive data transitions for a sustained length of time.

The maximum run length of consecutive "0" or "1" bits which must be tolerated depends on the protocol application and the data encoding defined for a given protocol. For example, protocols which use 8B/10B encoding are guaranteed to have no more than 5 bit times between data transitions. Protocols using another common encoding, 64B/66B, are guaranteed to see run lengths no longer than 66 bit times. Scrambled protocols may encounter much longer run lengths, and must determine requirements using statistical analysis. For example, Sonet/SDH is a scrambled protocol which specifies systems must meet a BER of 1×10^{-12}. It is generally accepted that run lengths of scrambled Sonet/SDH data longer than 80 bits statistically occur less frequently than the specified BER. Therefore, a Serdes used to receive Sonet/SDH data must tolerate a run length of 80 bits.

The run length which can be tolerated by a CDR design is related to the frequency tolerance between the two clock sources. In a system using plesiosynchronous clocks, the reference clock used by the receiver (and the CDR circuit) may be running at a slightly different frequency from the reference clock used by the transmitter, as is described further in Sect. 4.1.3.1. The frequency tolerance between the two clock sources is generally specified in parts per million (ppm). In a plesiosynchronous system, the CDR must continually correct the phase of the sample clock to remain in the center of the data eye. During periods where no data transitions are being received, the error in phase position builds up. Therefore, as the frequency tolerance of the system is increased (corresponding to larger allowed frequency difference between the clock sources), the run length which can be tolerated by the CDR design is reduced for a given performance (BER) target.

1.3.3.2 Clock Operation During System Initialization

In the above discussion, it was noted that some CDR architectures derive the sample clock from the received data using a PLL. During system initialization or during system operation when cables are unplugged, etc., no data transitions are received for a substantial period of time. For some Serdes, this results in clock outputs of the receiver changing frequency or stopping. Any downstream logic clocked by these clock outputs must be designed to be tolerant of this frequency change or to assume logic is not clocked during these periods.

1.3.4 Differential Driver

The differential driver stage is an analog circuit which drives the *true* and *complement* legs of the differential signal. Output data must be driven such that jitter is minimized. In some architectures, data is latched in a flip-flop clocked at the baud rate, and the output of this flop is driven onto the differential output. Such implementations require an internal high-speed clock running at the baud rate. This is illustrated in Fig. 1.16.

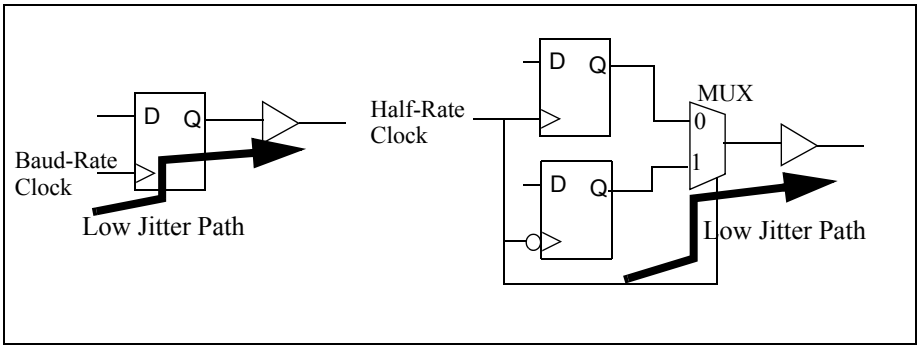


Fig. 1.16 Driver stage architectures

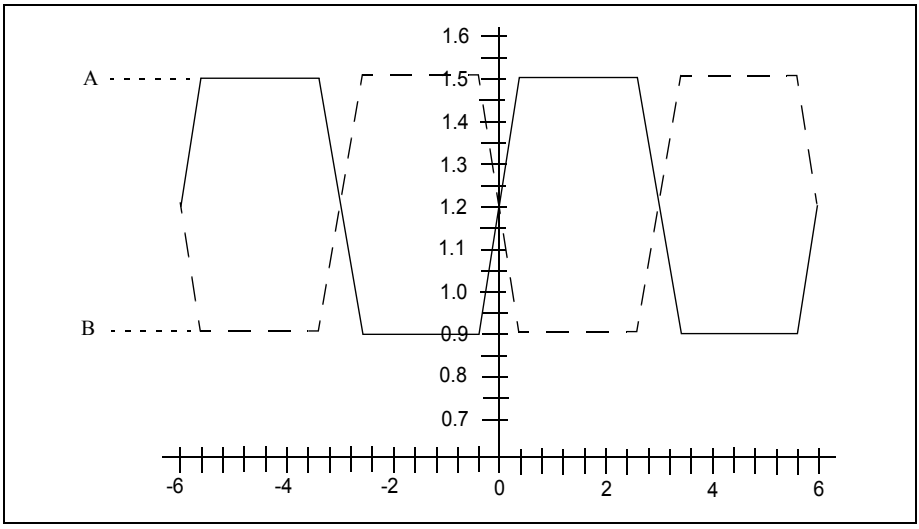


Fig. 1.17 Single-ended complementary signals

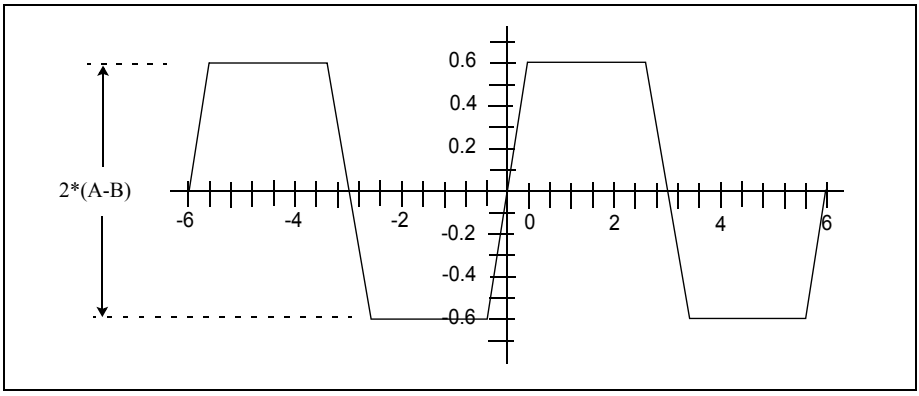


Fig. 1.18 Differential peak-to-peak signal

An alternative architecture, also shown in Fig. 1.16, uses an internal high-speed clock running at a frequency equal to half of the baud rate. Data is latched in two flip-flops on alternate edges of the high-speed clock. The high-speed clock also controls a multiplexor which alternately selects which of the flops drives the differential driver. Depending on the characteristics of the silicon technology, this architecture may result in lower jitter than the full-rate architecture.

Figure 1.17 illustrates typical voltage swings for the two legs of the differential signal, assuming a termination voltage of approximately 1.8 V. The average voltage on the signal is the *common mode voltage* (V_{cm}). For this example:

$V_{cm} = (1.5\,V + 0.9\,V) / 2 = 1.2\,V.$

The *differential voltage* (V_{diff}) is calculated by taking the voltage of the true leg and subtracting the voltage of the complement leg. Figure 1.18 illustrates the differential waveform corresponding to the single-ended signals from Fig. 1.17.

This differential voltage swings between the following limits:

$V_{diff} = 1.5\,V - \ \ 0.9\,V = +0.6\,V$

$V_{diff} = 0.9\,V - \ \ 1.5\,V = \ \ 0.6\,V$

This waveform has a total *peak-to-peak differential voltage* of $1.2\,V_{ppd}$. Note that the peak-to-peak voltage of the differential signal is twice the peak-to-peak voltage of either single-ended signal considered individually.

1.3.5 Differential Receiver

The differential receiver stage is an analog comparator circuit which compares the *true* and *complement* legs of the differential signal and outputs a "0" or "1" logic level based on the relative signal voltages. Differential receiver stages used with DFEs are linear amplifiers; the comparator circuit is incorporated into the DFE.

1.3.6 Diagnostic Functions

Additional logic is often incorporated into the transmitter and receiver designs to provide diagnostic capabilities for chip manufacturing test, circuit board manufacturing test, and system diagnostic tests. Typical functions include:

1. Pseudo random Bit Sequence (PRBS) Checker. PRBS sequences can be checked by comparing received data to the output of a local linear feedback shift register implementing the corresponding characteristic polynomial. Receiver devices often include a PRBS checker capable of checking one or more PRBS test patterns.

2. Loopback or Wrap Paths. Full duplex Serdes devices often provide the capability to wrap transmitter outputs to receiver inputs in order to self-check the functionality of the Serdes. Simplex cores do not have this capability,

although some simplex transmitters include a test receiver, and some simplex receivers include a test transmitter to perform self-test.

3. JTAG 1149.1 and JTAG 1149.6. These JTAG standards are used for manufacturing test of circuit boards, and require insertion of boundary scan cells on all chip I/O to support this testing. Since such logic cannot be inserted on high-speed I/O without impacting signal integrity, the Serdes core must provide appropriate hooks to drive differential outputs from boundary scan cells at the transmitter device, and sample inputs in boundary scan cells at the receiver device. JTAG 1149.6 expands the capabilities of JTAG 1149.1 to permit testing through decoupling capacitors and support independent testing of the true and complement legs of differential signals. JTAG 1149.1 and 1149.6 are covered in detail in Sect. 7.1.

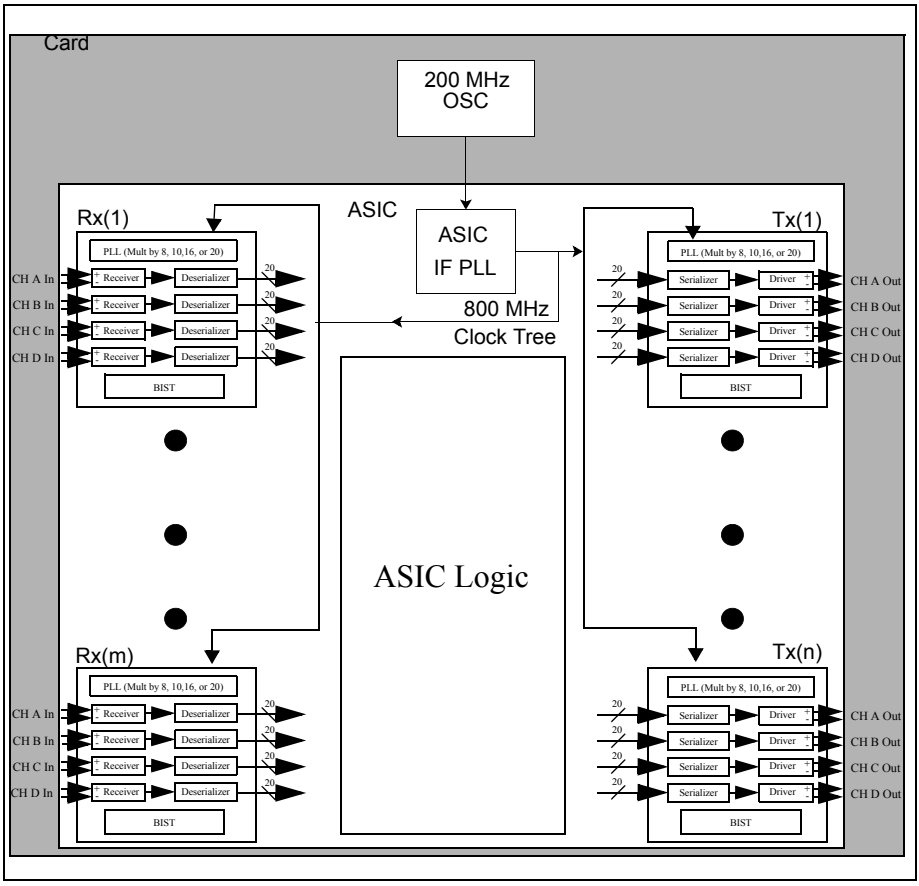


Fig. 1.19 Clock distribution example using an ASIC IF PLL