XAPP681 (v1.0) December 12, 2003

HD-SDI Receiver Using Virtex-II Pro RocketIO Multi-Gigabit Transceivers

Author: John F. Snow

Summary

The High-Definition Serial Digital Interface (HD-SDI) standard describes how to transport high-definition (HD) digital video serially over video coax cable. HD-SDI is used to connect HD video equipment in broadcast studios and video production centers. It is an evolution of the popular SDI standard that is widely used to transport standard-definition (SD) digital video in the broadcast industry.

The flexibility of the RocketIO™ multi-gigabit transceivers available in the Virtex-II Pro™ family devices combined with the programmable logic of the Virtex-II Pro FPGAs makes it possible to implement HD-SDI interfaces. Because every Virtex-II Pro FPGA has multiple RocketIO transceivers, multiple HD-SDI interfaces can be integrated into one Virtex-II Pro device along with other video processing functions.

This application note describes how to implement HD-SDI receivers. An HD-SDI receiver built in a Virtex-II Pro FPGA is presented as a reference design.

Introduction

Use of HD-SDI, defined by the SMPTE 292M standard, is increasing rapidly in broadcast studios and video production centers as the broadcast industry ramps up support for HDTV broadcasting [Ref 1].

HD-SDI builds upon the widely used SDI standard for transporting SD digital video. The older SDI standard is referred to as SD-SDI to differentiate it from HD-SDI. The SD-SDI and HD-SDI standards share the same electrical characteristics and encoding scheme. However, HD-SDI uses a higher bit rate to accommodate the higher bandwidth requirements of uncompressed HD digital video signals. Because SD-SDI and HD-SDI share common electrical characteristics, it is possible to build video equipment that can support both standards through a single connection.

This application note discusses the HD-SDI receiver. A companion application note, XAPP680, describes how to implement the HD-SDI transmitter [Ref 2]. Other existing Xilinx application notes cover the SD-SDI standard [Ref 3]. Forthcoming Xilinx application notes will describe how to use the RocketIO transceivers to implement multi-rate capable interfaces supporting both HD-SDI and SD-SDI [Refs 4, 5]. Another related application note describes the implementation of an HD digital video pattern generator [Ref 6].

The HD-SDI standard supports both coax cable and optical fiber interfaces. Coax cable has been the more popular of the two due to lower cost and commonality with SD-SDI. This application note only discusses the implementation details for the coaxial cable interface. However, since the data formats and encoding schemes for the optical interface option are identical to the coaxial interface option, the reference design presented in this application note is directly applicable to implementing an HD-SDI receiver with an optical fiber interface.

As of this writing, there is a proposal for a new standard, SMPTE 372M, defining a dual-link HD-SDI interface. This proposal uses two HD-SDI interfaces to provide twice the bandwidth, allowing higher bandwidth video formats to be supported. This proposed new standard is not specifically addressed in this application note, but the HD-SDI reference design described here can be used as a building block for implementing a dual-link HD-SDI interface.

HD-SDI Receiver Functions

This section describes the basic functions implemented by an HD-SDI receiver. Figure 1 is a block diagram of a typical HD-SDI receiver. Refer to the HD-SDI Data Format section of XAPP680 for a description of the video formats supported by HD-SDI and the details of the format of the HD-SDI bitstream.

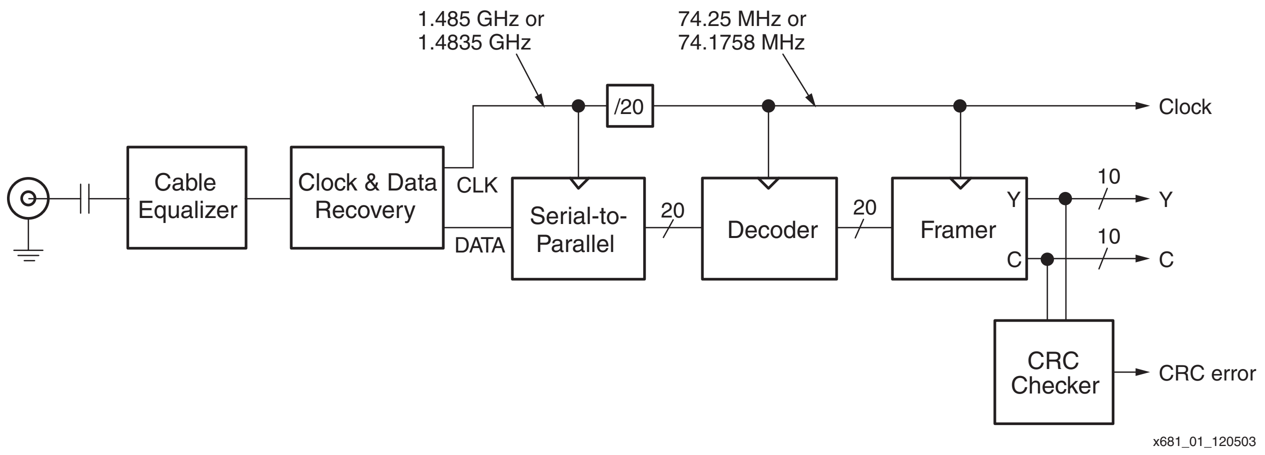


Figure 1: **HD-SDI Receiver Block Diagram**

Cable Equalization

HD-SDI uses two bit rates: 1.485 Gbps and 1.485 / 1.001 Gbps (approximately 1.4835 Gbps). The HD-SDI bitstreams are sent serially using an unbalanced (single-ended) driver over 75-ohm coaxial cable up to 100 meters in length.

The coax cable causes frequency-dependent attenuation of the signal, where the higher frequency components of the signal are attenuated more than the lower frequency components. The coax cable also causes frequency-dependent phase distortion, where the higher frequency components are phase shifted more than lower frequency components. After passing through 100 meters of coax cable, the HD-SDI signal will be severely distorted and attenuated. The receiver must compensate for this attenuation and distortion before attempting to recover the signal.

Cable length equalization is used to compensate for the attenuation and distortion introduced by the coax cable. The SMPTE 292M HD-SDI standard states that receivers typically work with an attenuation of 20 dB at one-half the clock rate. Because this is not a requirement, the standard permits HD-SDI receivers that cannot recover a signal with 20 dB of attenuation.

Typically, an adaptive cable length equalizer is used in HD-SDI receivers. Such an equalizer actively monitors the amount of attenuation and distortion present on the incoming signal and applies the correct amount of equalization to the signal. The cable length is allowed to change without requiring a change to the equalizer, as would be the case if fixed length equalization were used.

Clock and Data Recovery

After cable equalization, the HD-SDI receiver recovers the clock and data from the HD-SDI bitstream. This recovery typically is done with a PLL-based clock and data recovery (CDR) unit. A recovered clock usually is required for an HD-SDI receiver because the HD-SDI protocol has no provisions for clock correction to allow the incoming bitstream to be easily resynchronized to a local reference clock. Instead, the recovered clock from the CDR unit generally is used to clock all HD-SDI receiver logic downstream from the CDR unit.

When building an HD-SDI receiver using Virtex-II Pro devices, the RocketIO transceiver implements the CDR function and also deserializes the bitstream. The RocketIO transceiver provides a recovered clock that runs at the HD-SDI word rate (1/20th the bit rate). For HD-SDI, the recovered clock from the RocketIO transceiver runs at either 74.25 MHz or 74.25 / 1.001 MHz, depending on which bit rate is currently being received.

Decoding

As described in XAPP680, HD-SDI uses a two-stage encoding algorithm, where the first stage performs pseudorandom scrambling and the second stage performs non-return-to-zero (NRZ) to non-return-to-zero-inverted (NRZI) conversion. After recovering the data, the HD-SDI receiver must decode it by reversing the two encoding steps: first it converts the NRZI data to NRZ, and then it undoes the pseudorandom scrambling. Figure 2 shows conceptually how the HD-SDI bitstream is decoded in a serial manner.

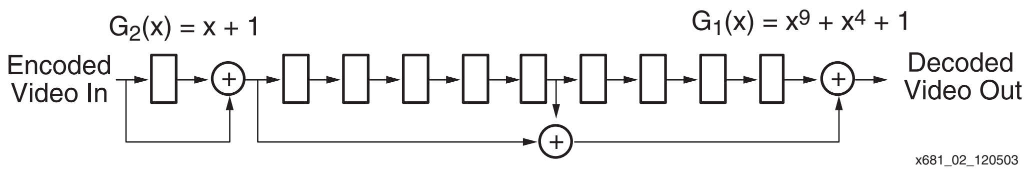


Figure 2: **HD-SDI Decoding Algorithm**

The RocketIO transceivers have built-in 8B/10B decoders. However, they do not have HD-SDI decoders. So, the recovered data from the RocketIO transceiver bypasses the decoding logic built into the RocketIO transceiver and is provided directly to the RXDATA port still encoded. The HD-SDI reference design described in this application note implements the HD-SDI decoder in the fabric of the Virtex-II Pro FPGA. The data is decoded in a parallel manner, 20 bits per clock cycle.

Framing

The recovered data words from the CDR unit and from the HD-SDI decoder are not word aligned. The CDR unit has no concept of where the video sample boundaries are in the continuous stream of incoming bits. The decoder does not care where the sample boundaries are since it can decode the data without this information. However, after decoding, it is necessary to identify the sample boundaries and realign the data so that each 20-bit sample is properly aligned and contains a 10-bit Y word and a 10-bit C word. This process of realigning the data is called *framing*.

The framer in the HD-SDI receiver monitors the incoming data and looks for the bit sequences that mark the beginning of the timing references. There are two timing references per video line: the end-of-active video (EAV) and the start-of-active video (SAV). Both the EAV and SAV have the same format and are four 10-bit words long. The first three words are always fixed values. The first word of the timing reference is a word of all ones and has a hex value of $3FF_H$. The second and third words of the timing reference are made up of all zeros (000_H). The fourth word of the timing reference is called the XYZ word. Figure 3 shows the format of the XYZ word of the timing reference. The sequence of 10 ‘1’ bits followed by 20 ‘0’ bits that marks the beginning of each timing reference is unique in the HD-SDI video stream and can occur only at the beginning of the timing reference.

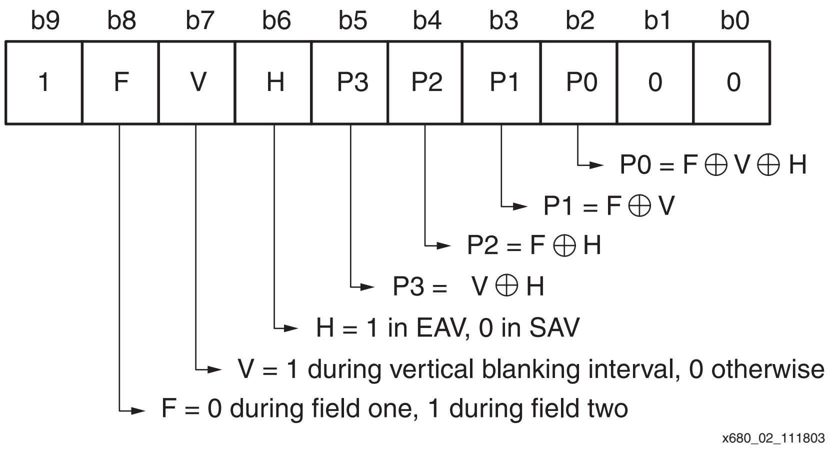


Figure 3: **XYZ Word Format**

HD-SDI divides the video stream into separate channels called the luma (Y) channel and the chroma (C) channel. Each channel has its own set of timing references. The channels are considered to be synchronous so that the first word of the EAV, for example, would appear on both the Y channel and the C channel at the same time.

Before transmission by the HD-SDI transmitter, the Y and C channels are interleaved so that a C word is transmitted first followed immediately by the corresponding Y word. Figure 4 shows the details of this interleaving.

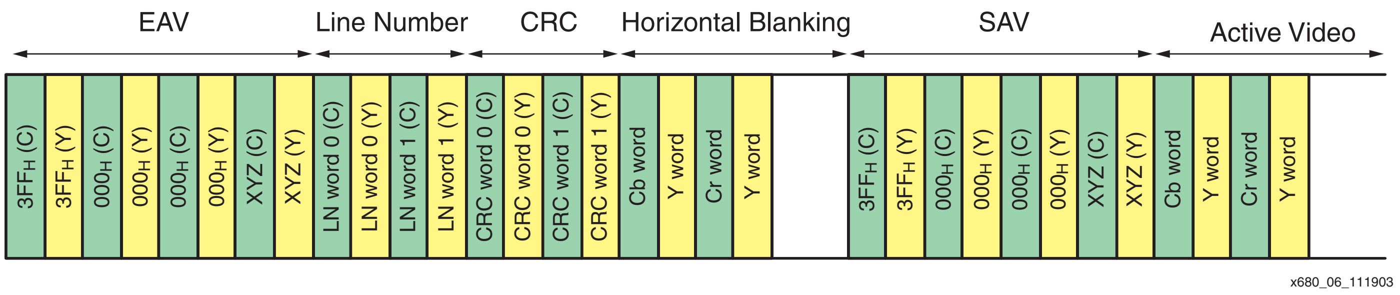


Figure 4: **Interleaved Data Stream**

The framer in the HD-SDI receiver must look for the unique $3FF_H$, 000_H, 000_H sequence that marks the beginning of a timing reference. Only this unique pattern in the HD-SDI bitstream can be used as a reference point for realigning the data. Due to the interleaving of the Y and C channels, the framer sees the following sequence for each timing reference: $3FF_H$, $3FF_H$, 000_H, 000_H, 000_H, 000_H.

The framer looks for this unique pattern beginning at any possible bit position in the recovered data coming from the HD-SDI decoder. Once this pattern is identified, the framer knows the bit offset of the least significant bit of each sample in the data words coming from the decoder. A barrel shifter is used to realign each 20-bit sample.

Figure 5 shows how a framer correctly aligns the data from the decoder. The data going into the framer is unaligned and contains an EAV beginning at bit 12. The 20 ‘1’ bits and 40 ‘0’ bits that make up the first three words of the interleaved EAVs are shown in red. The 20 bits of the two XYZ words are shown in blue. After the framer, the data is realigned so that the first bit of the EAV is positioned as the least significant bit of the C channel.

In the HD-SDI receiver reference design, the framer function is implemented in the fabric of the Virtex-II Pro FPGA.

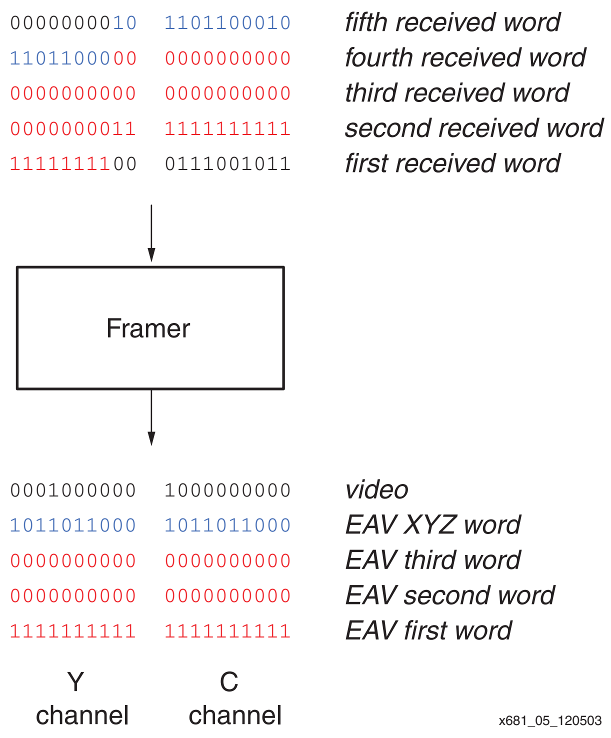

0000000010 1101100010 fifth received word
1101100000 0000000000 fourth received word
0000000000 0000000000 third received word
0000000011 1111111111 second received word
1111111100 0111001011 first received word
Framer
0001000000 1000000000 video
1011011000 1011011000 EAV XYZ word
0000000000 0000000000 EAV third word
0000000000 0000000000 EAV second word
1111111111 1111111111 EAV first word
Y channel
C channel
x681_05_120503

Figure 5: **Framer Example**

CRC Checking

After the HD-SDI video stream has been aligned by the framer, the receiver does cyclic-redundancy-code (CRC) checking to determine if any errors have occurred in the transmission of the data. Each video line contains an 18-bit CRC. The CRC is formatted into two 10-bit words located after the EAV of each line. The two words immediately after the XYZ word of the EAV contain the line number of the video line. The two words containing the CRC are located immediately after the line number words.

The Y and C channels each have their own CRCs. The receiver computes CRC values separately for both the Y and C channels as it receives a line of video. When the CRCs embedded in the video stream arrive, the receiver compares them to the CRCs that it has calculated. If the CRCs differ, an error has been detected.

XAPP680 has more details on how the CRCs are computed and formatted.

Additional HD-SDI Receiver Functions

After CRC checking, the basic functions of the HD-SDI receiver are complete. Depending on the application, the HD-SDI receiver also may perform some additional functions. Some of these additional functions are described here.

The HD-SDI receiver may examine the video stream to determine its video format. The HD-SDI standard supports many different video formats. There are two ways to determine the video format:

- by the characteristics of the video itself (word/line counting) or
- by the finding a special ancillary data (ANC) packet that identifies the video format.

The SMPTE 352M standard specifies an ANC packet that can be used to uniquely identify the format of the *video payload*. However, if the video stream does not contain an SMPTE 352M payload ID packet, the video format can be identified by counting the number of words per line and lines per frame in the video. An example of such a video format detector is included in the reference design.

It is useful to derive some video timing information from the received video stream. A video decoder can generate various video timing signals, such as horizontal and vertical blanking, from the timing reference signals. A more sophisticated video decoder might implement a *flywheel* which keeps track of where the timing reference signals are expected, repairs defective timing references, and inserts timing references when they are missing. A simple video timing decoder is included in the reference design.

The SMPTE 352M video payload ID is one type of ancillary data that can be included in the horizontal and vertical blanking intervals of the HD-SDI data stream. Another common use of ancillary data packets is to carry embedded digital audio. Some HD-SDI receivers might need to detect certain types of ancillary data packets and separate that ancillary data from the main video stream. The general format of ancillary data packets is given in the SMPTE 291M standard. Ancillary data packets are easy to detect in the video stream because they begin with a unique three-word sequence, similar to the first three words of the timing reference. The first three words of an ancillary data packet are 000_H, $3FF_H$, $3FF_H$.

HD-SDI Receiver Requirements

The SMPTE 292M document places a few requirements on the HD-SDI receiver. Basically, the receiver must be compatible with the single-ended, AC coupled electrical signal generated by the HD-SDI transmitter. The receiver must provide a 75-ohm impedance to the cable interface with a return loss of at least 15 dB from 5 MHz to 1.485 GHz.

The SMPTE 292M standard states that it is typical for HD-SDI receivers to receive signals attenuated by up to 20 dB. Because this is not a requirement of the standard, receivers that cannot recover the data when the input signal has been attenuated by 20 dB are permitted.

The SMPTE 292M standard contains a jitter template describing the maximum amount of jitter that can be produced by the HD-SDI transmitter. HD-SDI receivers should have input jitter tolerance exceeding the maximum allowed transmitter jitter as described by the jitter template, although not specifically required by the standard. The amount by which the receiver exceeds the jitter template is the jitter margin of the receiver.

Figure 6 shows the SMPTE 292M jitter template. The horizontal axis is jitter frequency, and the vertical axis is jitter amplitude given in UI(1). The output jitter of an HD-SDI transmitter must be below the template for every jitter frequency within the specification. An HD-SDI receiver should be able to tolerate more jitter than the transmitter is allowed to produce at each jitter frequency. When the input jitter tolerance of an HD-SDI receiver is plotted onto Figure 6 (blue line), all points of the plot should be above the jitter template line. For any particular jitter frequency, the vertical distance between the receiver's input jitter tolerance and the jitter template is the receiver's jitter margin at that frequency.

1. UI stands for Unit Interval. One UI is equal to the duration of one bit in the bitstream. For HD-SDI, one UI is about 673 ps.

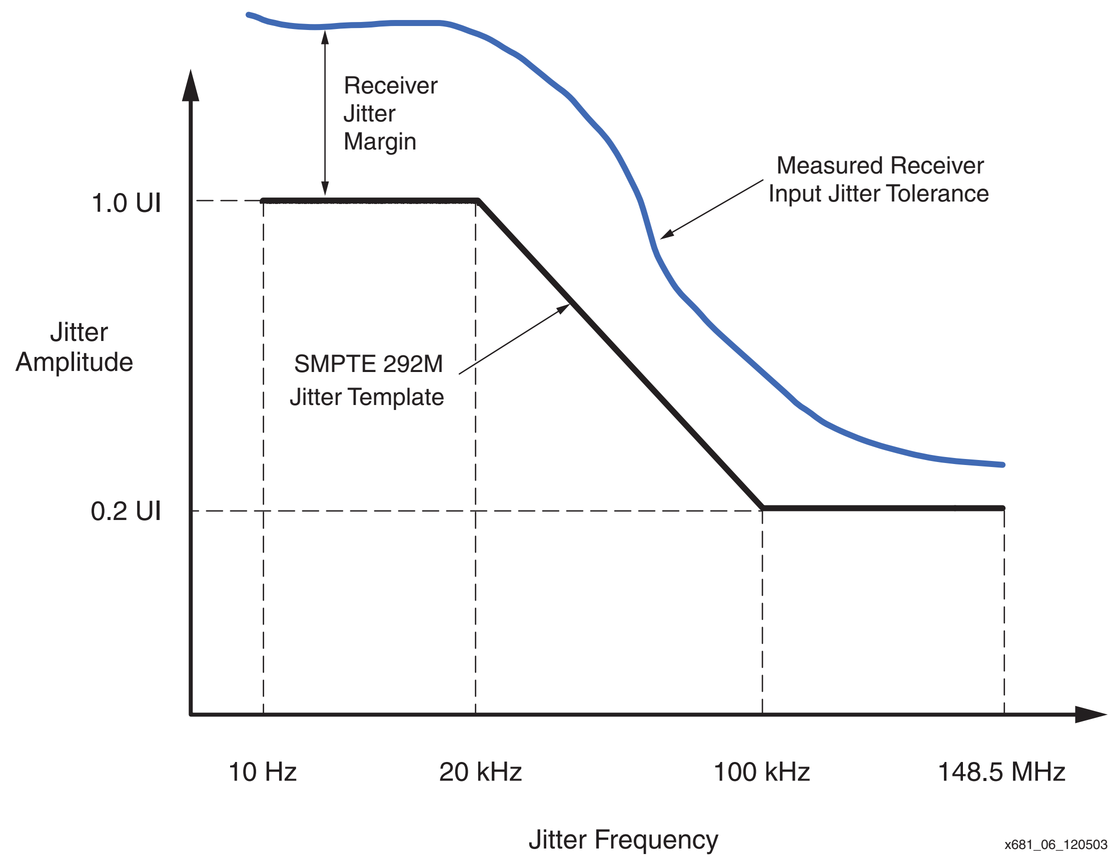


Figure 6: **SMPTE 292M Jitter Template**

SMPTE recommended practice RP 198 describes two worst-case *pathological* waveforms that can be produced by the HD-SDI encoder. One pathological waveform is poorly DC balanced and can cause problems with cable equalizers not designed to tolerate this waveform. Any cable equalizer designed specifically for HD-SDI use should be tolerant of this waveform. The second pathological waveform is essentially a low-frequency square wave consisting of 20 Low bits followed by 20 High bits. This square wave can repeat across the entire active portion of a video line. This low-frequency waveform can cause problems for the PLL inside of the CDR unit. The RocketIO CDR unit has been tested extensively with this waveform and is fully tolerant of it.

Implementing the HD-SDI Receiver

This section details how to implement an HD-SDI receiver using the RocketIO transceivers in Virtex-II Pro FPGAs. The reference design described here is implemented and tested on the Xilinx SDV Demo Board [Ref 7].

Cable Equalizer

As previously described, an HD-SDI receiver usually has an adaptive cable length equalizer to compensate for attenuation and distortion of the signal caused by long runs of coax cable. The RocketIO transceivers in the Virtex-II Pro FPGA do not include adaptive cable length equalizers. So, an external cable equalizer must be used to interface the HD-SDI cable to the RocketIO receiver. As a side benefit, the cable equalizer also converts the single-ended HD-SDI signal into a differential signal. The CML inputs of the RocketIO receiver require a differential input signal. Most HD-SDI cable equalizers currently available have 3.3V LVPECL outputs that are not directly compatible with the 2.5V CML inputs of the RocketIO transceiver. AC coupling is used to interface the LVPECL outputs of the cable equalizer to the CML inputs of the RocketIO transceiver. Figure 7 shows a typical AC coupled interface between a Gennum GS1524 cable equalizer and a RocketIO receiver.

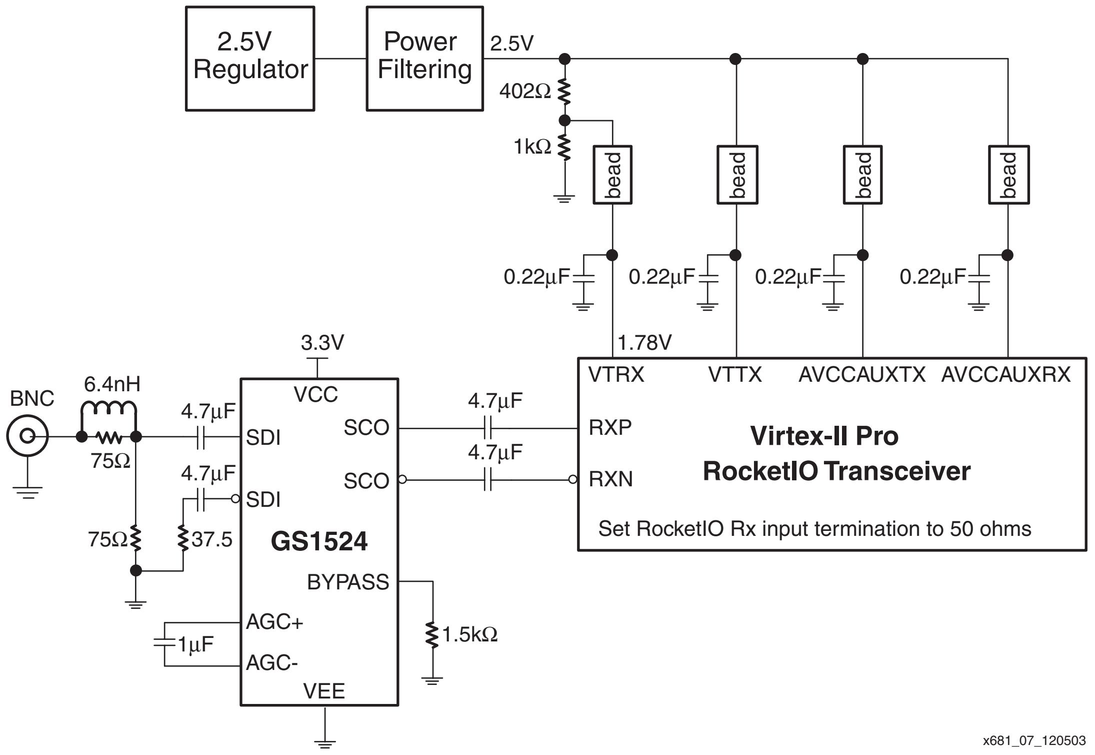


Figure 7: **Interfacing a Cable Equalizer to the RocketIO Receiver**

There are several important details in Figure 7:

- The recommendations given in the GS1524 data sheet [Ref 8] must be followed for the interface network between the BNC cable connector and the GS1524's input.
- The GS1524 is a multi-rate cable equalizer capable of supporting both HD-SDI and SD-SDI. HD-SDI only cable equalizers are also available.
- The coupling capacitors between the GS1524 and the RocketIO receiver must be in the 1 µF to 10 µF range to pass the HD-SDI pathological waveforms without too much voltage droop. Typically, 4.7 µF capacitors are used.
- The input impedance of the RocketIO receiver must be set to 50 ohms, and the circuit board traces between the equalizer and the RocketIO receiver must have an impedance of 50 ohms.
- As described in the *RocketIO Transceiver User Guide* [Ref 9], when using AC coupling, the RocketIO receiver termination voltage (VTRX) must be between 1.6V to 1.8V. As shown in the figure, the required termination voltage can be generated from 2.5V using a voltage divider network. The resistor values shown are sized to supply the termination voltage to a single RocketIO receiver, so this resistor network must be duplicated for each RocketIO receiver used as an HD-SDI receiver.

In rare cases, it might not be necessary to use a cable equalizer. For example, if the HD-SDI bitstream is always sent over a very short length of cable or a backplane within a chassis, the transmission path length may be short enough that cable equalization is not required. In such cases, the incoming single-ended HD-SDI signal must be converted to a differential signal compatible with the CML inputs of the RocketIO receiver, unless the HD-SDI transmitter can be designed to provide a differential signal. While a differential HD-SDI signal is not within the HD-SDI specification, it would provide a superior solution inside of a proprietary chassis, especially when a cable equalizer is not used.

RocketIO Transceiver Clocks

The RocketIO transceiver requires two types of clocks: *reference* clocks and *user* clocks. The reference clocks are used by the RocketIO receiver as a reference for the CDR PLL. The user clocks are used to clock data out of the RocketIO receiver into the fabric of the FPGA. In addition, the RocketIO receiver also produces a recovered clock, called RXRECCLK.

The following sections describe the clocking requirements of the RocketIO transceivers oriented towards implementing HD-SDI interfaces. More details about the clocking requirements of the RocketIO transceivers can be found in the *RocketIO Transceiver User Guide*.

Reference Clocks

The RocketIO transceiver uses reference clocks for two different purposes:

1. In the transmitter, the reference clock provides a low-jitter frequency reference that the transmitter multiplies by 20 to obtain the bit-rate clock for the transmitter's serializer.
2. On the receiver side, the reference clock is used to *spin up* the CDR circuit so that it can quickly lock to the bit rate of the incoming bitstream. The receiver's PLL does not operate properly without a reference clock or if the reference clock frequency is not close enough to the frequency of the HD-SDI bitstream.

The reference clocks are required to be 1/20th the frequency of the bitstream ±100 ppm. Because HD-SDI has two different bit rates (1.485 Gbps and 1.485 / 1.001 Gbps), the RocketIO receiver must have both 74.25 MHz and 74.25 / 1.001 MHz reference clocks available if it is to support both HD-SDI bit rates.

As described in detail in the *RocketIO Transceiver User Guide* and in XAPP680, each RocketIO transceiver has four reference clock inputs. A set of MUXes selects one of the four reference clock inputs as the active input. The method by which these MUXes are controlled limits dynamic switching to between two of the four inputs. Switching to the other set of two inputs requires reconfiguring the RocketIO transceiver. When implementing a RocketIO receiver, two reference clocks (either REFCLK and REFCLK2 or BREFCLK and BREFCLK2) typically are used, where one reference clock provides the 74.25 MHz reference frequency and the other provides 74.25 / 1.001 MHz.

Note that the selected reference clock is used by both the transmitter and the receiver in a RocketIO transceiver. It is not possible to select one reference clock for the transmitter and another reference clock for the receiver in a single RocketIO transceiver. However, different RocketIO transceivers can have different reference clocks. This sharing of the reference clock by the transmitter and receiver has significant implications when trying to implement an HD-SDI transmitter and receiver in the same RocketIO transceiver. This topic is discussed in detail in "Appendix B: Implementing an HD-SDI Receiver and Transmitter with one RocketIO Transceiver."

The reference clocks have fairly stringent jitter requirements. At HD-SDI bit rates, the reference clock inputs should have no more than 100 ps of peak-peak jitter. Xilinx recommends the use of low-jitter oscillators with differential outputs to provide the reference clocks for the RocketIO transceivers. The transmitter section of the RocketIO transceiver actually imposes the most stringent requirements for low jitter on the reference clocks because jitter on the reference clock becomes jitter on the transmitter's output. The RocketIO receiver is less sensitive to reference clock jitter. HD-SDI receivers have been successfully tested on the SDV board using singled-ended clock sources for the reference clocks. See "Appendix C: A Low-Cost Reference Clock Solution" for details.

User Clocks

The user clocks clock data out of the HD-SDI receiver and into the fabric of the FPGA. Each RocketIO transceiver requires two user clocks on the receiver side called RXUSRCLK and RXUSRCLK2. Each RocketIO transceiver also has two user clocks for the transmitter side called TXUSRCLK and TXUSRCLK2. If the transmitter portion of the transceiver is not used, the TXUSRCLK and TXUSRCLK inputs should still be driven with valid clock signals. In this case, simply connect TXUSRCLK to RXUSRCLK and TXUSRCLK2 to RXUSRCLK2.

RXUSRCLK is the clock signal that clocks data out of the RocketIO transceiver. The receiver output ports, such as RXDATA, change synchronously with the rising edge of RXUSRCLK. The frequency of RXUSRCLK is equal to the word rate of the HD-SDI interface, either 74.25 MHz or 74.25 / 1.001 MHz.

The frequency and phase relationships between RXUSRCLK and RXUSRCLK2 depend on the width of the RXDATA port of the RocketIO transceiver. For HD-SDI, a 20-bit RXDATA port is convenient to use because it matches the data word width of HD-SDI (10 bits of Y and 10 bits of C). When using a 20-bit RXDATA port, RXUSRCLK2 must have the same frequency and phase as RXUSRCLK (simply connect RXUSRCLK and RXUSRCLK2 to the same clock source). Consult the *RocketIO Transceiver User Guide* for RXUSRCLK2 requirements when other RXDATA port widths are used.

Note that the RocketIO transceiver's RXDATA port is actually only 16 bits wide when a two-byte interface is selected. However, with the internal 8B/10B decoder bypassed, four additional output data bits are provided on other receiver output ports to form a 20-bit output data word. For simplicity, this application note calls the entire 20-bit output port RXDATA.

In serial protocols that have clock correction capability, the RXUSRCLK and RXUSRCLK2 signals usually are derived from the same source as the reference clock. Then the RocketIO transceiver's clock correction capability is used to occasionally insert or remove idle characters to compensate for the minor differences between the actual clock frequency of the incoming bitstream and the frequency of the local reference clock.

HD-SDI does not support clock correction. Therefore, deriving RXUSRCLK and RXUSRCLK2 from the reference clock quickly results in either an overflow or underflow condition on the output data port of the RocketIO receiver because RXUSRCLK and RXUSRCLK2 are running at a slightly different frequency than the frequency of the incoming bitstream.

When implementing an HD-SDI receiver, the recovered clock (RXRECCLK) from the RocketIO receiver is used as the source of RXUSRCLK and RXUSRCLK2. When connected in this manner, as shown in Figure 8, RXUSRCLK and RXUSRCLK2 always run at the same frequency as the incoming bitstream provided the RocketIO receiver's CDR unit is locked to the bitstream. Thus underflow and overflow conditions are prevented on the RXDATA port of the RocketIO receiver.

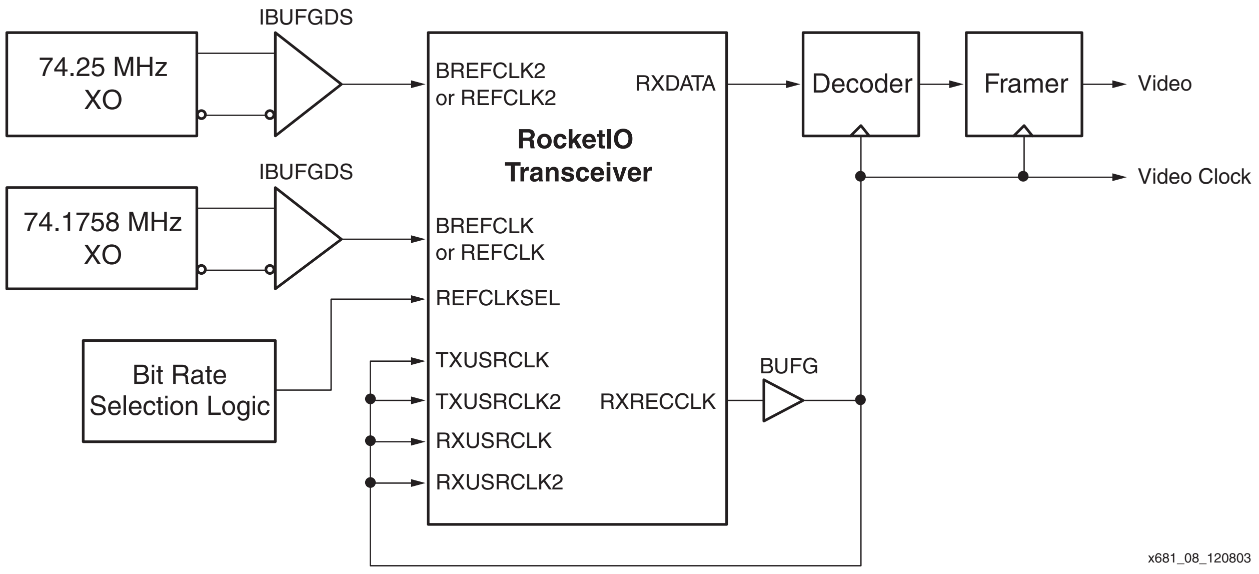


Figure 8: **Typical RocketIO Transceiver Clock Connections for an HD-SDI Receiver**

RXRECCLK

The RXRECCLK output of the RocketIO transceiver is the recovered clock from the receiver's CDR unit. When the CDR unit is locked to the incoming bitstream, this clock is exactly 1/20th the frequency of the bitstream. When the CDR unit is not locked to the bitstream frequency, RXRECCLK runs at the same frequency as the selected reference clock. Thus RXRECCLK always provides a word-rate clock for the HD-SDI receiver logic downstream from the RocketIO transceiver.

As shown in Figure 8, it is common to connect RXRECCLK to a BUFG global clock buffer. The output of the BUFG can be connected to the RXUSRCLK and RXUSRCLK2 inputs of the RocketIO transceiver and also to the clock inputs of the other portions of the HD-SDI receiver, such as the decoder, framer, and CRC checker.

Reference Design

The HD-SDI receiver reference design can be downloaded from http://www.xilinx.com/bvdocs/appnotes/xapp681.zip. A high-level description of the reference design is in the following section. Detailed information about the reference design can be found in "Appendix A: Reference Design Details."

Most of the HD-SDI receiver reference design is contained in the module called hdsdi_rx. This module contains the HD-SDI decoder, framer, CRC checker, and the video format detector. It does not include the RocketIO transceiver module. The RocketIO module is kept separate from hdsdi_rx so that the RocketIO transceiver can be shared between an HD-SDI transmitter and receiver, if desired. Also not included in the hdsdi_rx module are the video timing decoder module (hdsdi_rx_timing) and a module (hdsdi_rx_autorate) that automatically toggles between the reference clock inputs to the RocketIO transceiver until the HD-SDI receiver locks.

An example of how to connect the hdsdi_rx module to the RocketIO transceiver and the other modules is given in the sdv_hdsdi_rx module. This HD-SDI receiver example was designed specifically for the Xilinx SDV demo board. On the SDV board, there are no provisions for bringing the received parallel video out of the Virtex-II Pro FPGA. So, the received video is simply checked for CRC errors to determine correct reception. Figure 9 is a block diagram of the sdv_hdsdi_rx design.

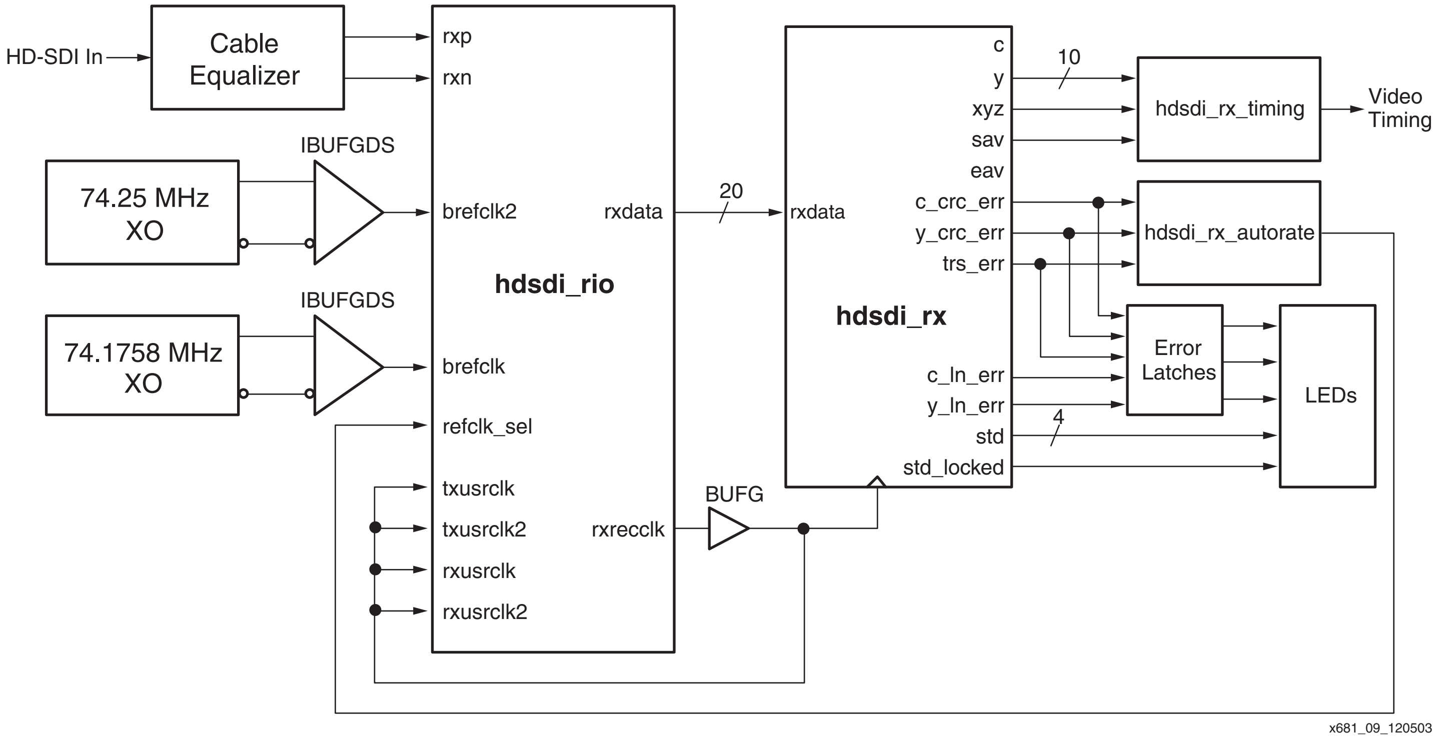


Figure 9: **Xilinx SDV Demo Board HD-SDI Receiver Reference Design**

Figure 10 shows a block diagram of the main HD-SDI receiver module, hdsdi_rx. This module contains the four submodules described in the following paragraphs.

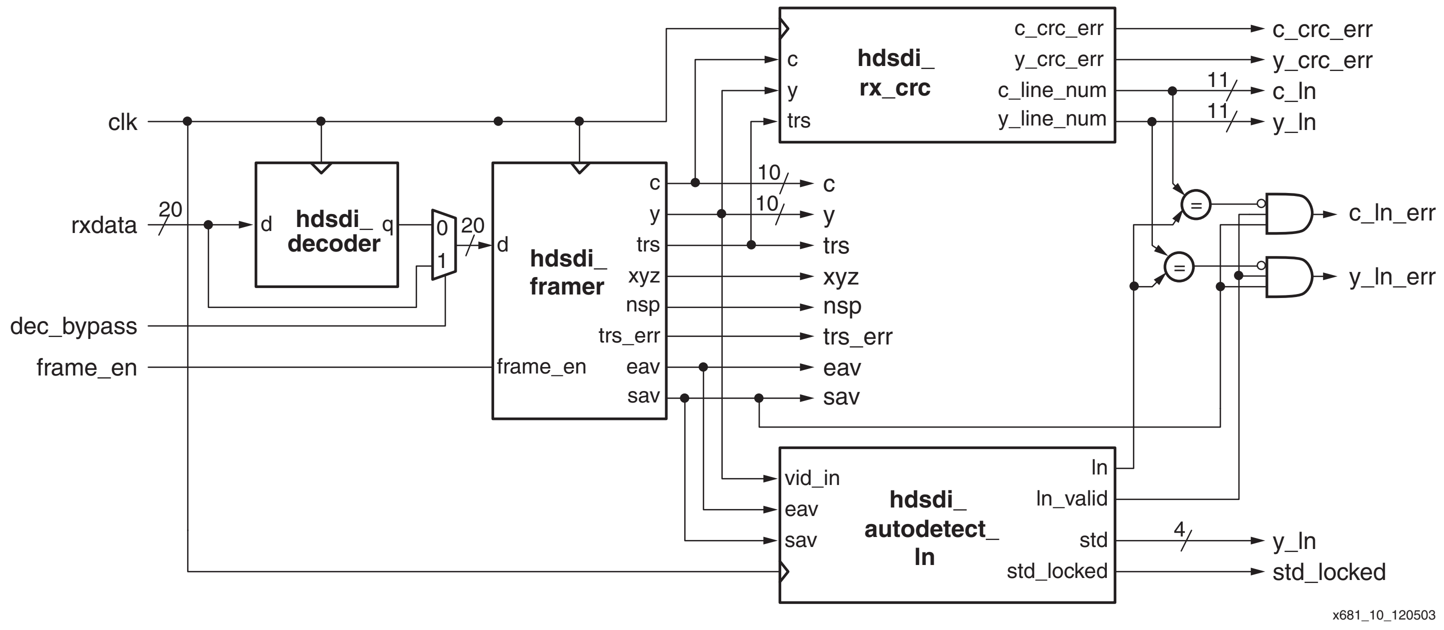


Figure 10: **hdsdi_rx Block Diagram**

The 20-bit parallel data comes into the hdsdi_rx module from the RocketIO transceiver. The data is encoded and unframed at this point. The data words are sent into the hdsdi_decoder module. The decoder performs the two-step decoding process.

The output of the decoder is connected to the input of the framer module. There are two different framer modules provided in the reference design. They perform the same framing function but are implemented using different resources in the FPGA. The module called hdsdi_framer implements all framer functions using the fabric of the FPGA (LUTs and flip-flops). The hdsdi_framer_mult module provides an alternative implementation where six MULT18X18 multiplier blocks implement the barrel shifter function inside the framer, reducing the amount of FPGA fabric required for the framer by about one-half. This implementation can be a good trade-off if the multiplier blocks otherwise are unused.

The hdsdi_rx_crc module computes CRC values for both Y and C channels for each video line and compares them to the CRCs inserted into the video stream by the HD-SDI transmitter.

Finally, the hdsdi_autoformat_ln module examines the data and determines the video format. Once it recognizes the format, it asserts the std_locked output and outputs a 4-bit code indicating the format. This module is identical to the module of the same name described in detail in XAPP680.

Figure 11 shows the block diagram of the hdsdi_rio module. This module is a wrapper around the RocketIO transceiver primitive (GT_CUSTOM). In addition to the RocketIO primitive, the module contains bit swap functions on the input and output data ports of the RocketIO primitive and a reset delay circuit for the RocketIO transceiver.

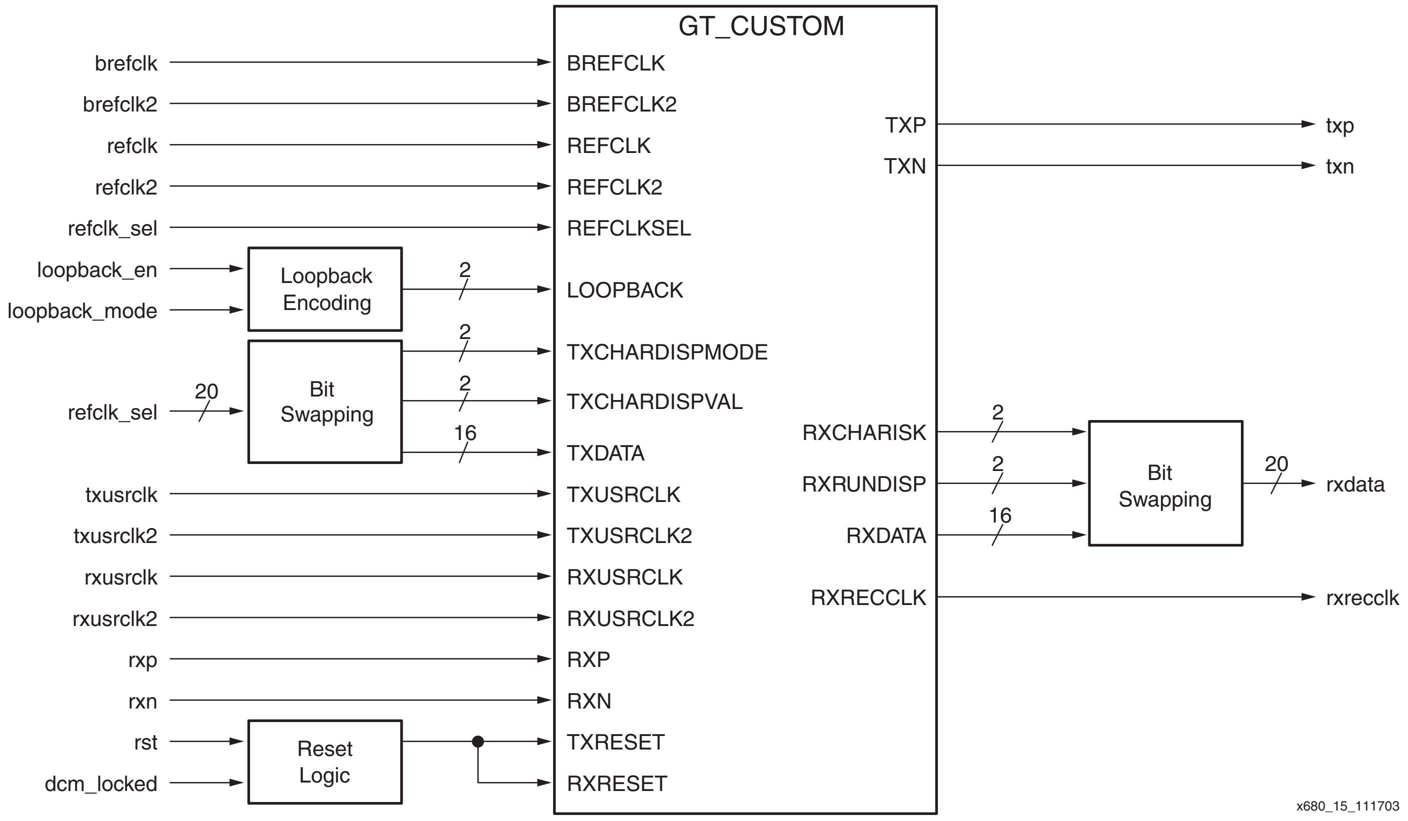


Figure 11: **hdsdi_rio Module**

The RocketIO transceiver transmits the MSB of the TXDATA port first. Likewise, the RocketIO receiver outputs the first bit it receives on the MSB of the RXDATA port. HD-SDI always transmits the LSB first, just the opposite of how the RocketIO transceiver operates. In order to accommodate this difference, the hdsdi_rio module swaps the bit order on the input and output ports.

The RXRESET input of the GT_CUSTOM primitive must remain asserted for at least two RXUSRCLK cycles after all clock inputs become stable. The hdsdi_rio primitive contains some logic to keep the RXRESET input asserted until several clock cycles after the dcm_locked input becomes asserted. This input is called dcm_locked because it can be driven by the LOCKED output of a DCM, if a DCM is used to generate the RXUSRCLK and TXUSRCLK signals. If a DCM is not used to generate these clock signals, then the dcm_locked input either can be connected to another appropriate signal that indicates when the clocks are stable or can be tied High if the clocks are always running.

When an HD-SDI bitstream initially is connected to the RocketIO transceiver, the selected reference clock might not be the correct reference clock for the frequency of the bitstream. If the correct reference clock is selected, then the RocketIO transceiver quickly locks to the bitstream, and the HD-SDI receiver begins decoding and framing the data. However, if the wrong reference clock is selected, the HD-SDI receiver receives the video with many errors. The hdsdi_rx_autorate module examines the errors detected by the HD-SDI receiver and determines when it is appropriate to change the frequency of the reference clock. This module is described in more detail in "Appendix A: Reference Design Details."

Figure 12 shows the results of input jitter tolerance measurements made on an HD-SDI receiver design implemented on the Xilinx SDV demo board. The input jitter tolerance of the receiver was measured at different jitter frequencies and then plotted relative to the HD-SDI transmitter jitter template. As can be seen in the figure, the receiver's input jitter tolerance is well above the HD-SDI jitter template for all jitter frequencies measured.

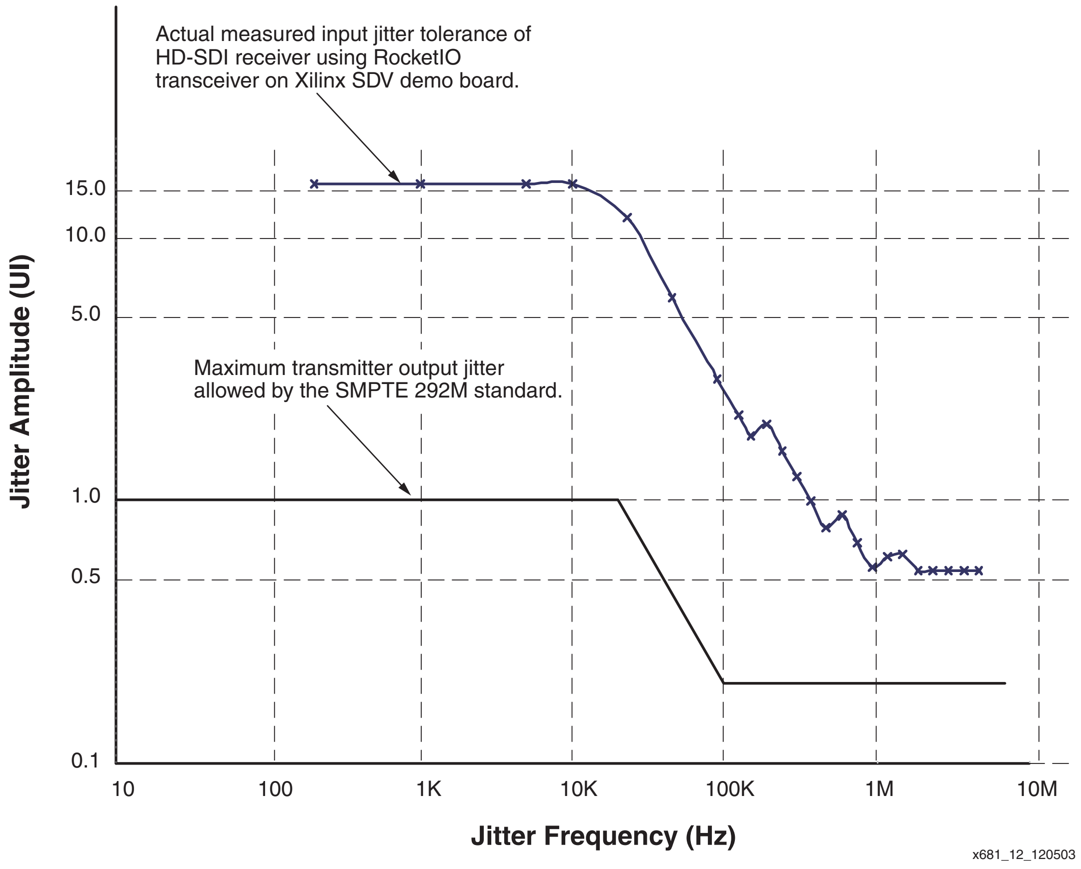


Figure 12: **HD-SDI Receiver Input Jitter Tolerance**

Reference Design Size

Table 1 shows the FPGA resources used by the HD-SDI receiver reference design. The complete reference design with all the modules is shown with the regular hdsdi_framer and hdsdi_framer_mult modules. The sizes of the optional modules (hdsdi_rx_timing, hdsdi_autodetect_ln, and hdsdi_rx_autorate) are also shown. If the optional modules are not included, their sizes can be subtracted from the implementation size of the fully featured design.

Table 1: **Reference Design Implementation Sizes**

Design	LUTs	FFs	MULT18X18s
HD-SDI receiver with all features using hdsdi_framer	581	361	0
HD-SDI receiver with all features using hdsdi_framer_mult	477	361	6
Size of optional hdsdi_rx_timing	16	12	0
Size of optional hdsdi_autodetect_ln	174	71	0
Size of optional hdsdi_rx_autorate	30	21	0

Conclusion

This application note describes the implementation details of an HD-SDI receiver using the RocketIO multi-gigabit transceivers available in the Virtex-II Pro FPGA family. An HD-SDI receiver easily can be implemented from RocketIO transceivers combined with an HD-SDI decoder, framer, and other support functions built in the fabric of the FPGA.

The HD-SDI receiver reference design requires very few resources in the FPGA, making it quite easy to implement multiple HD-SDI interfaces in even the smallest member of the Virtex-II Pro family or to integrate video processing functions and an HD-SDI receiver all in the same part.

References

The following references provide related information for this application note:

1. All the SMPTE standards referenced in this application note are available from The Society of Motion Picture and Television Engineers. These standards can be purchased at the SMPTE website: http://www.smpte.org.
2. Xilinx application note XAPP680 – HD-SDI Receiver Using Virtex-II Pro RocketIO Multi-Gigabit Transceivers
3. The Xilinx SD-SDI applications notes are:
   - XAPP247 – Serial Digital Interface (SDI) Physical Layer
   - XAPP288 – Serial Digital Interface (SDI) Video Decoder
   - XAPP298 – Serial Digital Interface (SDI) Video Encoder
   - XAPP299 – Serial Digital Interface (SDI) Ancillary Data and EDH Processors
   - XAPP625 – SDI: Video Standard Detector and Flywheel Decoder
4. Xilinx application note XAPP683 – Multi-Rate HD/SD-SDI Transmitter using Virtex-II Pro RocketIO Transceivers (planned for future release)
5. Xilinx application note XAPP684 – Multi-Rate HD/SD-SDI Receiver using Virtex-II Pro RocketIO Transceivers (planned for future release)
6. Xilinx application note XAPP682 – HDTV Video Pattern Generator (planned for future release)
7. The Xilinx SDV Demo board is available from Cook Technologies (part number CTXIL103). Further information is available at http://www.cook-tech.com.
8. The Gennum GS1524 data sheet is located at http://www.gennum.com/vb/pdffiles/14976DOC.pdf
9. UG024: *RocketIO Transceiver User Guide*

Appendix A: Reference Design Details

This appendix contains detailed design information for the hdsdi_decoder, hdsdi_framer, and hdsdi_framer_mult modules.

hdsdi_decoder

The hdsdi_decoder module implements the two-stage decoding process to convert encoded HD-SDI data into decoded video data. The output data from the decoder is unaligned to word boundaries and must be framed by the hdsdi_framer module. Figure 13 shows a block diagram of the decoder module.

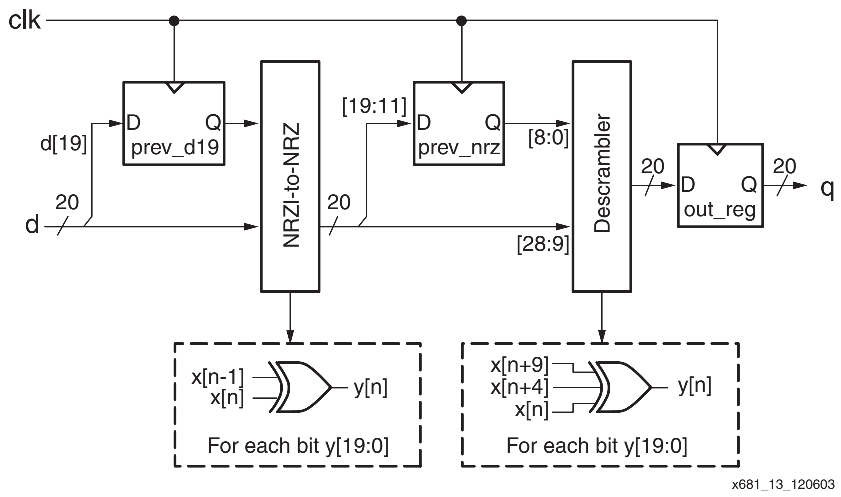


Figure 13: **hdsdi_decoder Block Diagram**

The decoder module first does the NRZI-to-NRZ conversion by XORing each bit with the previous bit in the bitstream. Remember that the LSB was the first bit received, so XORing d[1] with d[0] produces a new d[1] bit that has been converted to NRZ. The bit preceding d[0] is d[19] from the previous clock cycle. The prev_d19 register always captures the d[19] bit every clock cycle so that it can be XORed with the d[0] bit of the next clock cycle to produce an NRZ d[0] bit.

Then the 20 NRZ bits are passed through the descrambler block. This block XORes each bit with two other bits to reverse the pseudorandom scrambling done by the HD-SDI encoder. In order to descramble all 20 bits, the descrambler needs the nine most significant bits produced by the NRZI-to-NRZ converter during the previous clock cycle. These bits are held in the prev_nrz register.

hdsdi_framer

The data from the RocketIO receiver is not aligned and usually contains bits from two different samples. The framer realigns the video samples so that each sample output from the framer contains the Y and C words from the same sample.

The framer searches for the bit pattern that marks the beginning of a timing reference, either EAV or SAV. When this unique pattern is located, the framer knows the offset of the least significant bit of the sample within the 20-bit data from the decoder. The framer uses this offset value to control a barrel shifter to realign all the subsequently received video samples.

Figure 14 shows a block diagram of the hdsdi_framer module. The framer has three main sections: the input pipeline registers, the timing reference signal (TRS) detector, and the barrel shifter.

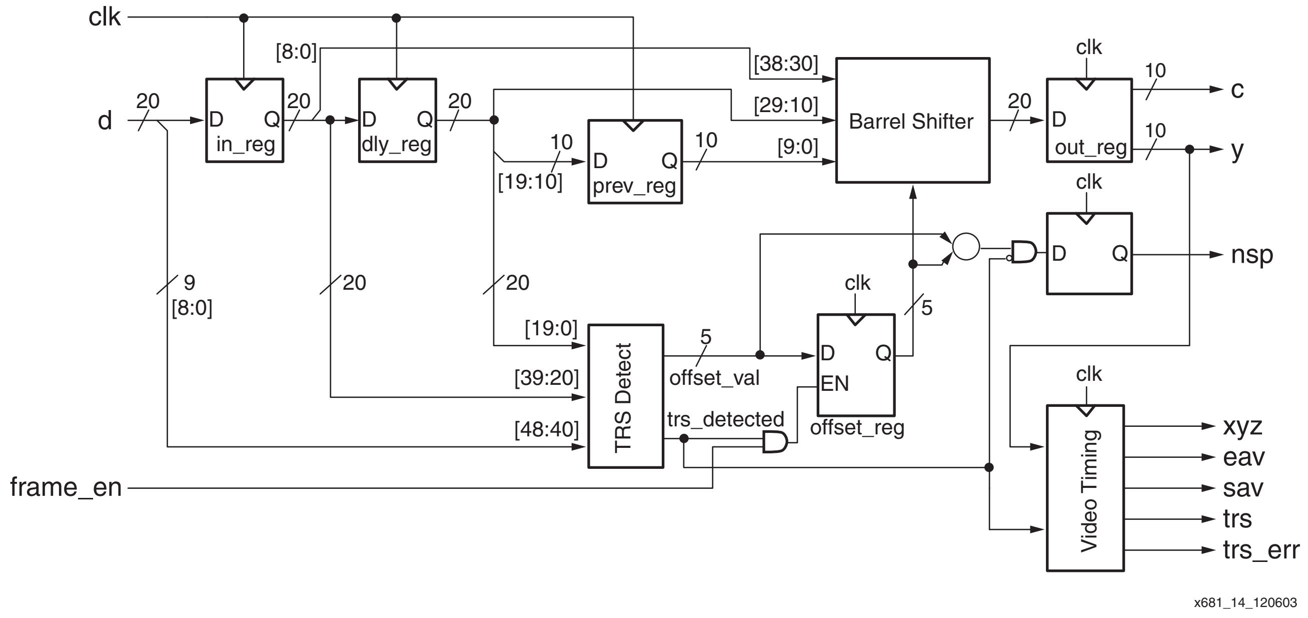


Figure 14: **hdsdi_framer Block Diagram**

A TRS (either EAV or SAV) begins with the unique sequence of words: $3FF_H$, 000_H, 000_H. The Y and C channels each have a TRS that occurs at the same time in both channels. So the framer sees a 60-bit sequence like this: $3FF_H$, $3FF_H$, 000_H, 000_H, 000_H, 000_H. The TRS detector does not need to match the entire 60-bit sequence. It can still look for the 30-bit sequence $3FF_H$, 000_H, 000_H because this pattern is still unique in the bitstream even after interleaving the channels. In this case, the $3FF_H$ and the second 000_H are from the Y channel, and the middle 000_H is from the C channel. Searching for just the 30-bit sequence, rather than the entire 60-bit sequence, significantly reduces the size of the TRS detector.

The 30-bit TRS sequence that the TRS detector is searching for can begin at any bit position in the 20-bit words coming into the framer. The TRS detector must look at bit 0 plus the next 29 bits for a match. It must also look at bit 1 plus the next 29 bits for a match, and so on up to bit 19 plus the next 29 bits. When the TRS detector takes in a new 20-bit word to determine if a TRS begins in this word, it must have this 20-bit word plus the 29 bits that were received immediately after it, forming a 49-bit vector that the TRS detector scans.

To form this 49-bit vector for the TRS detector, there are two pipeline delay registers called in_reg and dly_reg. Every clock cycle, in_reg captures the 20-bit word coming in the d input port, and dly_reg loads the previous word from in_reg. The 40 bits from these two registers plus the least significant nine bits from the d input port form the 49-bit vector scanned by the TRS detector.

An additional 10-bit pipeline register, called prev_reg, loads the most significant 10 bits of dly_reg every clock cycle. The TRS detector ignores the first $3FF_H$ word in the 60-bit TRS sequence and determines where the TRS begins by finding the starting location of the second $3FF_H$ word in the sequence. Thus, when the TRS detector finds the TRS location, some or all of the 10-bit C word that began the TRS sequence might have passed out of dly_reg. These bits are captured in prev_reg and are provided to the barrel shifter so that it has all the bits it needs to output the aligned sample.

The TRS detector implements a brute force pattern matcher, basically consisting of twenty 30-bit comparators. Each comparator is wired to look for the $3FF_H$, 000_H, 000_H pattern. There is one comparator for each of the 20 possible starting positions where the pattern could begin in the input vector.

When a new TRS is detected, the TRS detector asserts the trs_detected signal and generates a binary code, called offset_val, indicating the bit position where the TRS was detected. This code is compared with the bit offset currently being used by the framer, stored in offset_reg. If the frame_en input to the framer is asserted, then a difference between the new offset_val and offset_reg causes offset_reg to load the new offset_val. If frame_en is not asserted, then offset_reg is not loaded, and the nsp output is asserted, indicating that a TRS was detected that did not match the current offset used by the framer.

The nsp output, combined with the framer_en input, can be used by control logic external to the framer module to implement simple or sophisticated TRS filtering. Sometimes, noise corrupts the HD-SDI bitstream and produces a bit sequence in the bitstream that looks like a TRS. If the framer always aligns to new TRS offsets, such an erroneous TRS could cause the framer to misalign data until the next TRS arrives. By controlling the framer_en input and monitoring when new TRS starting positions are detected based on the nsp output, control logic can prevent the framer from switching to a new TRS offset until some number of timing reference sequences arrive at the new offset.

The framer keeps the nsp output asserted until either the framer is allowed to reload offset_reg by the assertion of frame_en or when a TRS is detected that has a starting position that matches the current contents of offset_reg.

A simple TRS filtering scheme can be implemented by connecting nsp to frame_en. With this connection, the framer does not reload offset_reg when a TRS is detected that does not match offset_reg. Instead, nsp is asserted. With the assertion of nsp, frame_en is asserted, and the framer is allowed to reload offset_reg when the next TRS is detected. This next TRS either:

- matches the current offset_reg value if the TRS that caused the assertion of nsp was erroneous, thus filtering out the erroneous TRS, or
- forces the offset_reg to reload if the new TRS does not match the current offset_reg contents.

The offset value stored in offset_reg controls a barrel shifter. This barrel shifter realigns the video samples to their correct word alignment. The input vector for the barrel shifter is 39 bits wide made from the 10-bit output of prev_reg, 20 bits from dly_reg, and the nine least significant bits of in_reg. Depending on the value of offset_reg, the barrel shifter extracts a 20-bit output value from the 39-bit input vector.

In the hdsdi_framer module, the barrel shifter is made from three levels of MUXes. The first level consists of 2:1 MUXes that shift the input vector either 0 or 16 bit positions. The second level takes the output of the first level and shifts it 0, 4, 8, or 12 bit positions. Finally, the third level takes the output of the second level and applies the final 0, 1, 2, or 3 bit position shift.

The output of the barrel shifter is loaded into the barrel_out register, which drives the y and c output ports of the framer.

The framer module also contains some decoding logic that produces some TRS-related video timing signals. The trs output is asserted when all four samples of a TRS are output from the framer. The xyz output is asserted when the XYZ word of a TRS is output from the framer. The eav and sav outputs are asserted when the framer outputs the XYZ word of an EAV or SAV, respectively. Finally, the trs_err output is asserted when the XYZ word is output from the framer, if an error is detected in the XYZ word by examining the XYZ protection bits.

hdsdi_framer_mult

The hdsdi_framer_mult module is an alternate implementation of the framer. It is identical in function to hdsdi_framer with the only difference in how the barrel shifter is implemented. In the hdsdi_framer_mult module, six MULT18X18 multiplier blocks implement the barrel shifter rather than LUTs, as in hdsdi_framer. This can be a good trade-off if the multipliers are not required for other purposes since it essentially reduces the number of LUTs required to implement the framer in half.

A MULT18X18 block can be used as a barrel shifter by inputting the data to be shifted into one of the multiplier's inputs and by putting a unary bit shift code into the other multiplier input. To shift the data zero positions, the shift code should be 1. To shift one position to the left, the shift code should be 2, and so on.

In the hdsdi_framer_mult module, the barrel shifter is implemented in two levels with three MULT18X18 blocks used in each level as shown in Figure 15. The top level of the barrel shifter shifts the input vector either 0 bit positions or 12 bit positions. The bottom level of the barrel shifter shifts the output of the top level from 0 to 11 bit positions. Thus, the barrel shifter can shift the input vector anywhere from 0 to 33 bit positions. However, the framer only requires shifts of 0 to 19 bit positions. So, the barrel shifter is not fully wired to support shifting by more than 19 bits.

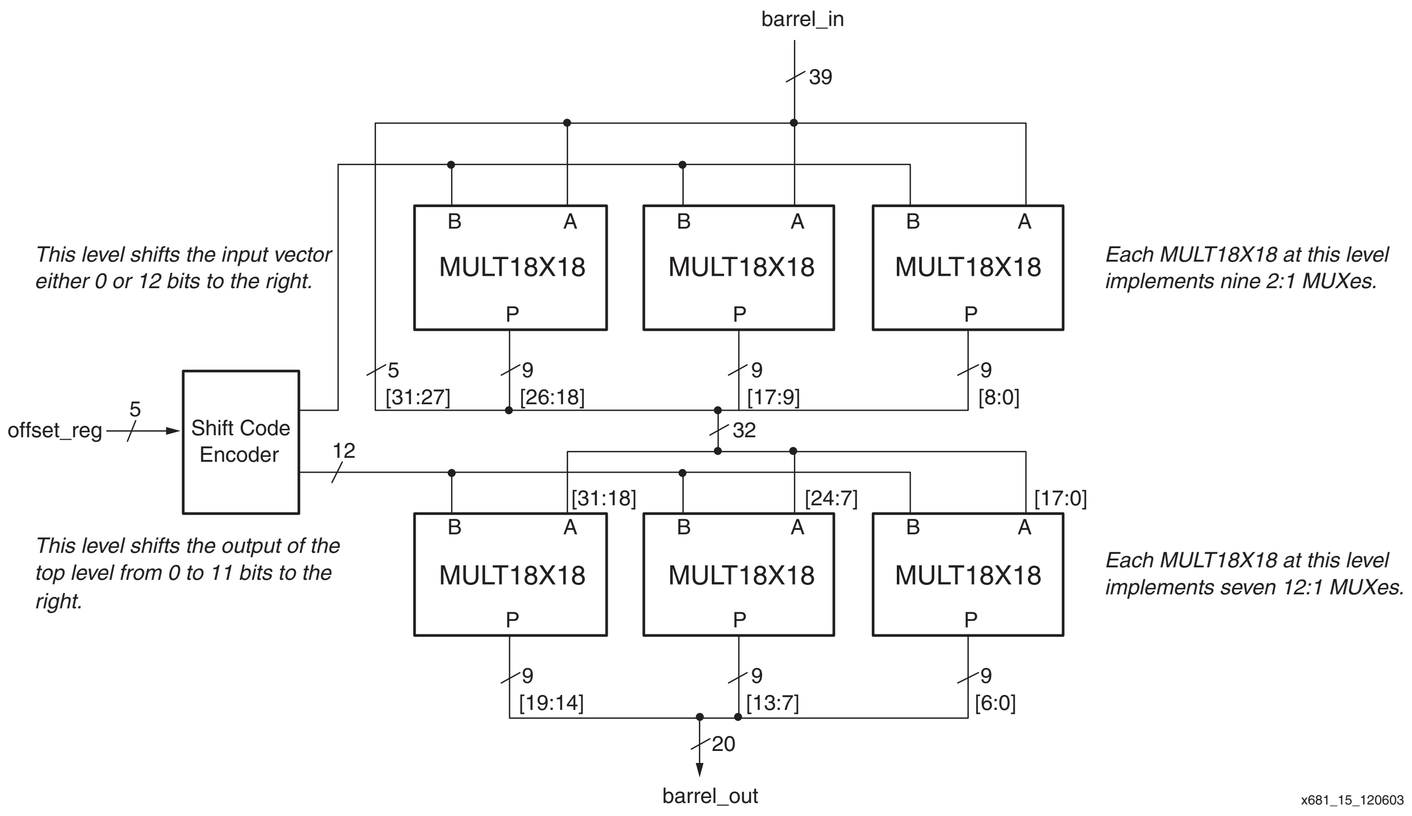

barrel_in
39
B
A
MULT18X18
P
This level shifts the input vector either 0 or 12 bits to the right.
Each MULT18X18 at this level implements nine 2:1 MUXes.
5
[31:27]
9
[26:18]
9
[17:9]
9
[8:0]
offset_reg
5
Shift Code Encoder
12
32
[31:18]
[24:7]
[17:0]
This level shifts the output of the top level from 0 to 11 bits to the right.
Each MULT18X18 at this level implements seven 12:1 MUXes.
[19:14]
[13:7]
[6:0]
20
barrel_out
x681_15_120603

Figure 15: **hdsdi_framer_mult Barrel Shifter**

In the top level of the barrel shifter, each multiplier acts like a 9-bit 2:1 MUX as shown in Figure 16. Note how every other output of the multiplier is used. The shift code, applied to the B input of the multiplier, only takes on the values of 1 or 2. When the shift code is 1, the X input bit of each MUX is selected and passes straight down to the output. When the shift code is 2, the Y input bit of each MUX is selected (by shifting it left one bit position).

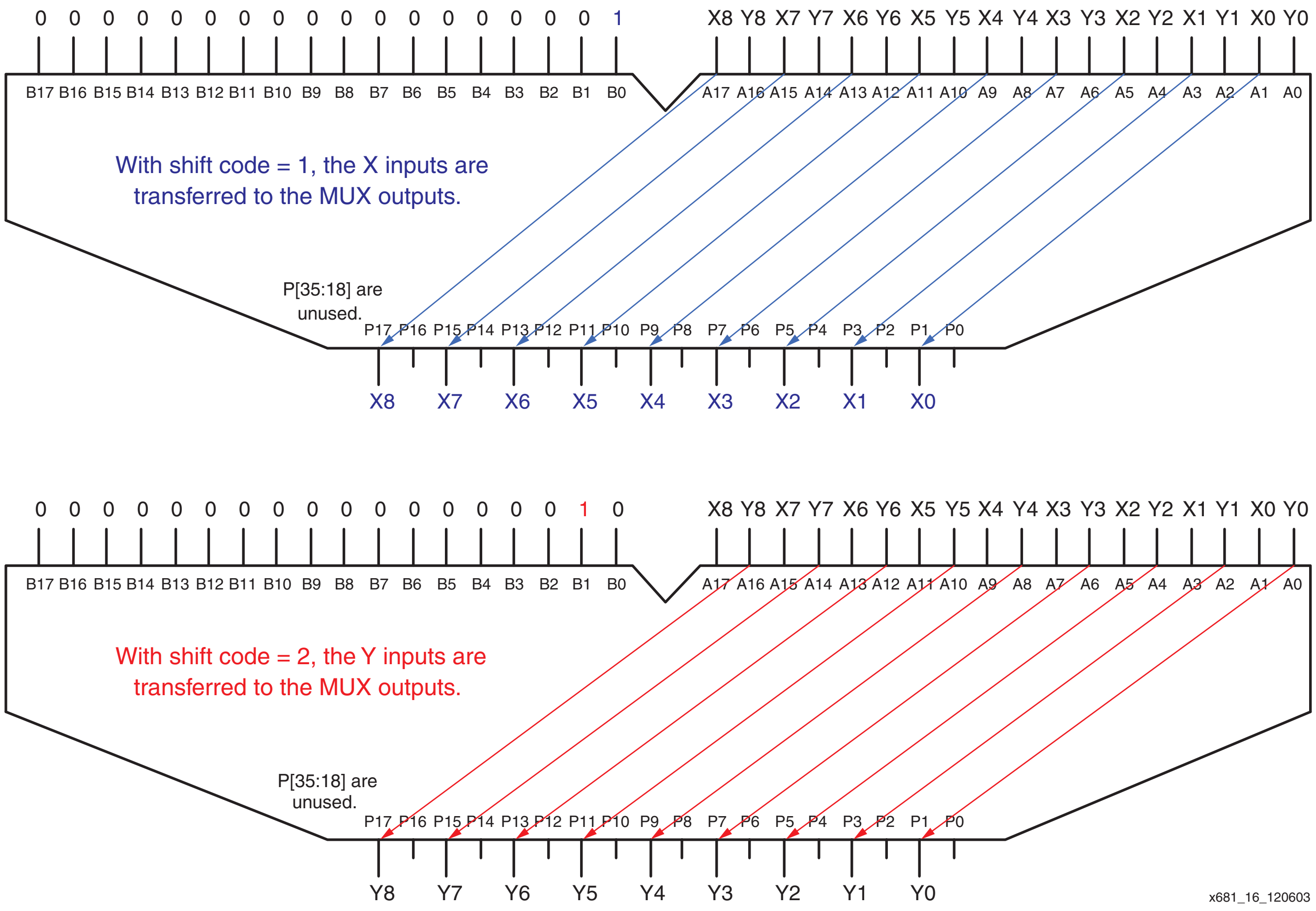
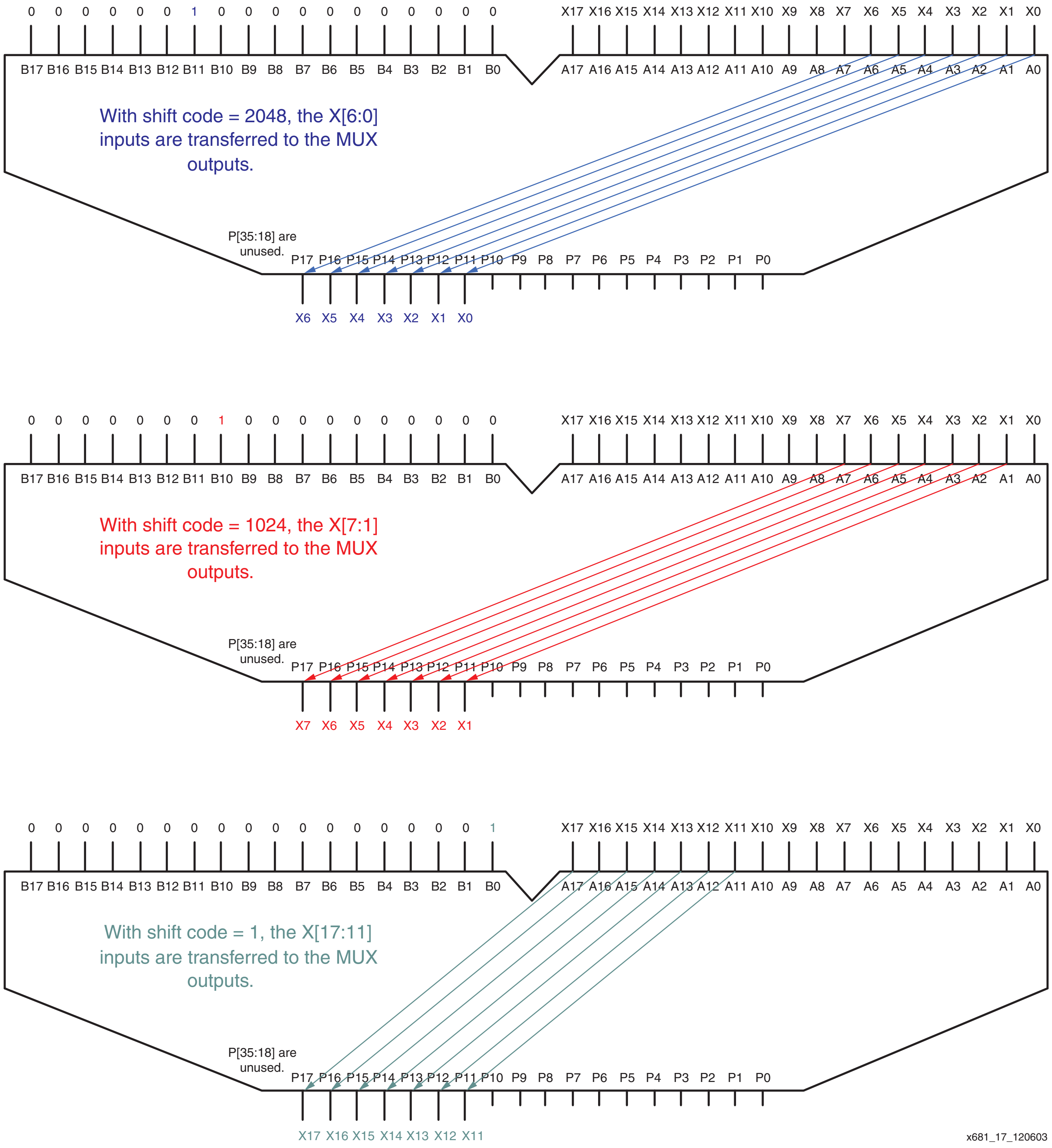

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
X8 Y8 X7 Y7 X6 Y6 X5 Y5 X4 Y4 X3 Y3 X2 Y2 X1 Y1 X0 Y0
B17 B16 B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
With shift code = 1, the X inputs are transferred to the MUX outputs.
P[35:18] are unused.
P17 P16 P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
X8 X7 X6 X5 X4 X3 X2 X1 X0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
X8 Y8 X7 Y7 X6 Y6 X5 Y5 X4 Y4 X3 Y3 X2 Y2 X1 Y1 X0 Y0
B17 B16 B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
With shift code = 2, the Y inputs are transferred to the MUX outputs.
P[35:18] are unused.
P17 P16 P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
x681_16_120603

Figure 16: **MULT18X18 used as Nine 2:1 MUXes**

In the bottom level of the barrel shifter, each multiplier acts like a 7-bit barrel shifter. Each multiplier has an 18-bit input vector connected to its A input and a shift code applied to its B input. If the shift code is 2048, the 7-bit output of the multiplier is equal to A[6:0]. If the shift code is 1024, the 7-bit output is equal to A[7:1], and so on until the shift code is 1 and the output is equal to A[17:11].



B17 B16 B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
X17 X16 X15 X14 X13 X12 X11 X10 X9 X8 X7 X6 X5 X4 X3 X2 X1 X0
P17 P16 P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
P[35:18] are unused.
With shift code = 2048, the X[6:0] inputs are transferred to the MUX outputs.
X6 X5 X4 X3 X2 X1 X0
With shift code = 1024, the X[7:1] inputs are transferred to the MUX outputs.
X7 X6 X5 X4 X3 X2 X1
With shift code = 1, the X[17:11] inputs are transferred to the MUX outputs.
X17 X16 X15 X14 X13 X12 X11
x681_17_120603

Figure 17: **MULT18X18 used as Seven 12:1 MUXes**

hdsdi_rx_autorate

This module selects the correct reference clock for the RocketIO transceiver. If the incorrect reference clock is selected, not matching the bitstream frequency, the RocketIO transceiver does not correctly recover the data.

Because the two HD-SDI frequencies are so close, much of the data is recovered correctly, even when the wrong reference clock is selected. The CDR unit in the RocketIO transceiver attempts to lock to the bitstream and recovers the data. However, after a certain period of time, the CDR unit determines that the frequency of the bitstream is not close enough to the frequency of the reference clock, and it switches back to using the reference clock to lock the CDR's PLL. Once locked to the reference clock, the PLL then is freed to lock to the bitstream. This cycle repeats continuously since the reference clock is not close enough to the bitstream frequency.

When the PLL is close to the bitstream frequency, valid data is received. When the PLL is locked to the reference clock, the data is invalid.

The hdsdi_rx_autorate module examines the errors received by the HD-SDI receiver and determines when to toggle the REFCLKSEL input to the RocketIO transceiver. The CRC checkers in the HD-SDI receiver do a very good job of detecting errors in the received data. However, looking at CRC errors alone is not sufficient because the CRC checkers only work correctly if the TRS symbols are found by the receiver. When the wrong reference clock is selected, TRS symbols are detected roughly two-thirds of the time. Because these symbols are found so often, simply looking for some number of consecutive missing TRS symbols is not an adequate strategy.

Thus, the hdsdi_rx_autorate module uses a combination of missing TRS symbols and CRC errors to determine when to toggle the reference clock. A state machine looks for some number of video lines containing either CRC errors or a missing SAV. When this error threshold is reached, the state machine toggles the reference clock and again begins monitoring the HD-SDI output.

The error threshold is controlled by a counter called errcnt. The bit width of this counter is controlled by the parameter or constant ERRCNT_WIDTH. The maximum number of lines containing errors before the switching threshold is reached is controlled by MAX_ERRS. By default, the error counter is two bits wide, and the switching threshold is reached when three consecutive video lines containing errors have been detected. By changing the error counter width and maximum error threshold, you can trade off error tolerance for switching latency. The more consecutive lines with errors that are required before the switching threshold is reached will decrease the likelihood that the reference clock will be switched erroneously. However, this also increases the latency for the reference clock switch when the frequency of the bitstream does change.

Figure 18 shows the state diagram of the state machine in hdsdi_rx_autorate.

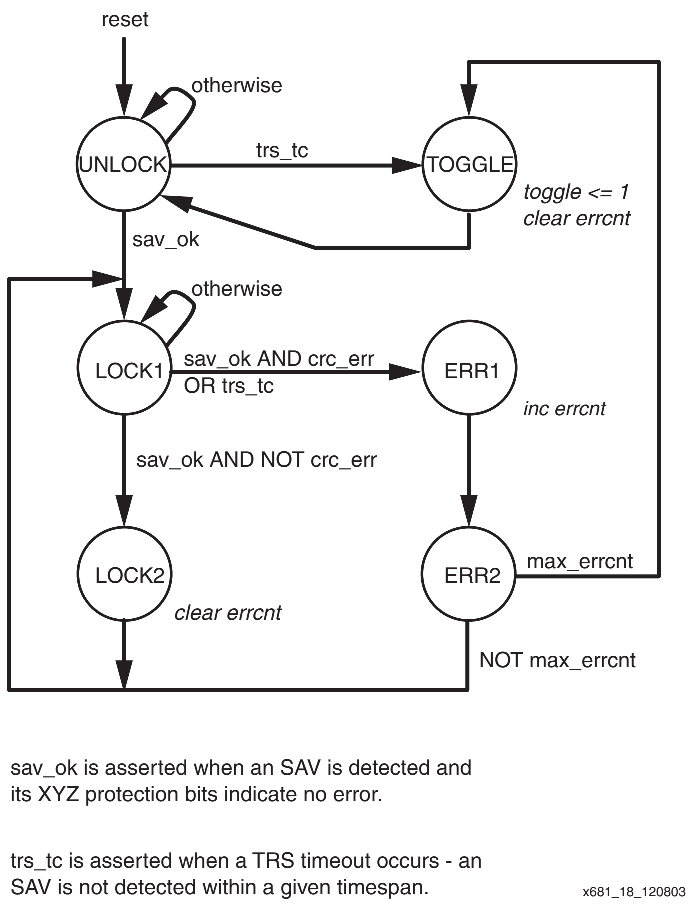


Figure 18: **hdsdi_rx_autorate State Diagram**

Appendix B: Implementing an HD-SDI Receiver and Transmitter with one RocketIO Transceiver

In the RocketIO transceiver, the receiver and transmitter portions of the transceiver share the same reference clock. It is not possible to select one reference clock for the transmitter and a different reference clock for the receiver inside of the same RocketIO transceiver. This sharing of the reference clock can lead to complications when trying to implement an HD-SDI transmitter and receiver in the same RocketIO transceiver. Depending on the application, it might not be possible to place the transmitter and receiver in one RocketIO transceiver.

If it is required that the transmitter is running at one HD-SDI bit rate while the receiver is receiving a bitstream at the other HD-SDI bit rate, then the receiver and transmitter must be placed in separate RocketIO transceivers.

One case where it makes sense to put the HD-SDI transmitter and receiver in the same RocketIO transceiver is a *pass-through* HD-SDI interface. In a pass-through interface, the video received by the HD-SDI receiver is connected to the HD-SDI transmitter to be retransmitted. In such a configuration, the transmitter always runs at the same rate as the receiver. Some processing might be done on the video between the receiver and the transmitter.

Figure 19 shows an example where the data is received by the RocketIO transceiver. The video is decoded and framed by the HD-SDI receiver, then a logo “bug” is inserted into the video. New CRC values for the video lines are calculated, the video is encoded by the HD-SDI transmitter, and the encoded video is transmitted by the RocketIO transceiver.

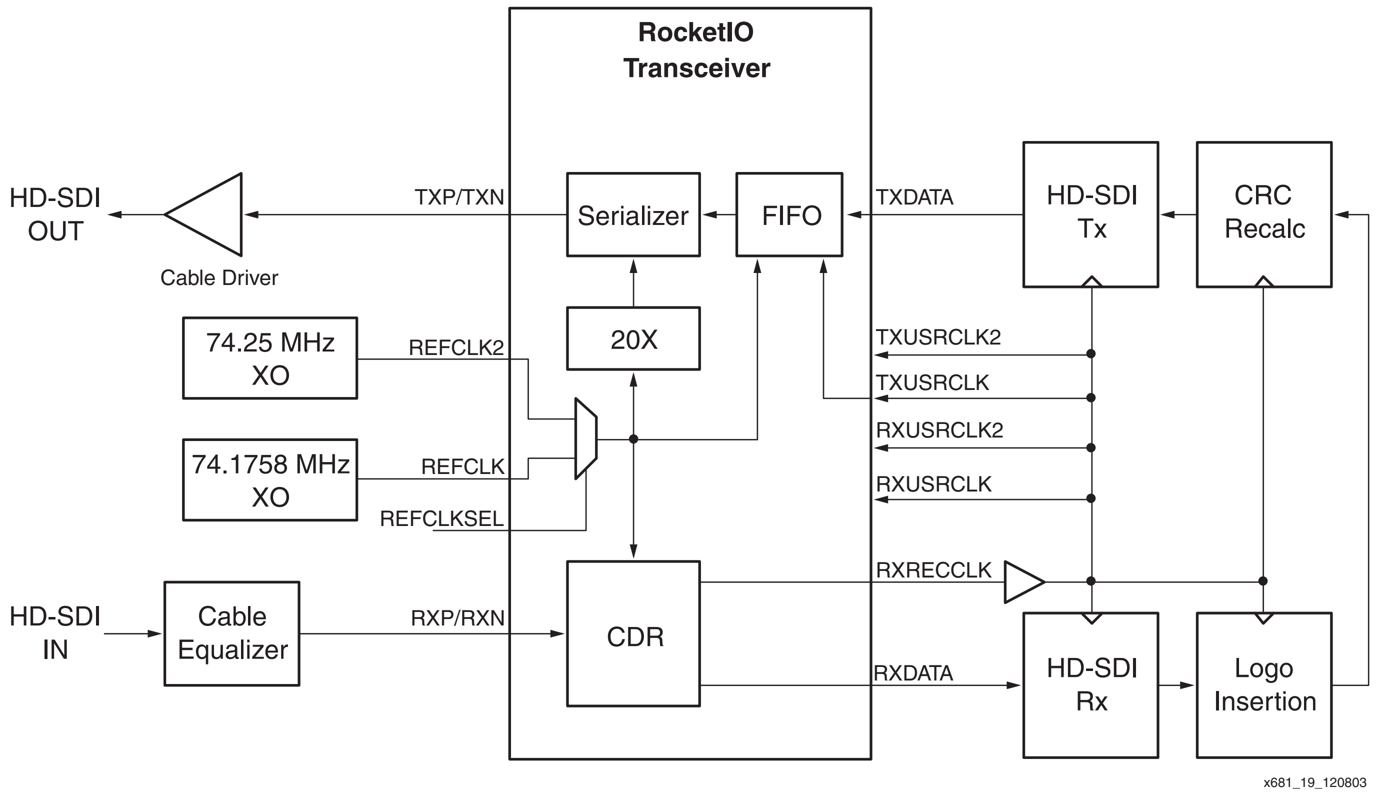


Figure 19: **The Problem with Sharing REFCLK Between Transmitter and Receiver**

As shown in Figure 19, the RXRECCLK signal from the RocketIO receiver clocks all logic downstream from the receiver because RXRECCLK is exactly the frequency of the video stream. RXRECCLK clocks the HD-SDI transmitter and also clocks the encoded data into the RocketIO transmitter because it is connected to TXUSRCLK and TXUSRCLK2. The FIFO at the input of the RocketIO transmitter moves the data from the TXUSRCLK domain to the REFCLK domain. Data is clocked from the TXDATA port into the FIFO using TXUSRCLK (which is connected to RXRECCLK). Data is read from the FIFO using the selected reference clock. Data is read from the FIFO using the reference clock because the serializer is running synchronous to the reference clock, but at 20 times the frequency of the reference clock.

Herein lies the problem with sharing the reference clock between the receiver and the transmitter. If TXUSRCLK is not frequency locked to the reference clock, then the FIFO at the input of the RocketIO transmitter underflows or overflows. However, when using one RocketIO for both the transmitter and receiver in a pass-through HD-SDI interface, TXUSRCLK must run at the video rate, that is, it must be connected to RXRECCLK. Because RXRECCLK and the reference clock are rarely the same frequency, the RocketIO transmitter FIFO will underflow or overflow in this configuration. There are two possible solutions to this problem.

The first solution is to resynchronize the video to the reference clock after it is received. Video resynchronization, sometimes done on a per line basis, usually is done on a frame basis. Resynchronizing video to the local reference clock requires a line buffer for line synchronization or a frame buffer for frame synchronization. Discussion of these video synchronization techniques is beyond the scope of this document.

A second solution is to somehow force REFCLK to track the frequency of RXRECCLK. RXRECCLK cannot be directly used as the reference clock because it has too much jitter.

Figure 20 shows a single RocketIO transceiver pass-through configuration that has been successfully tested on the Xilinx SDV demo board. The reference clocks are provided by VCXOs. The frequency of each VCXO is controlled by a phase detector, where the combination of the VCXO, the loop filter, and the phase detector forms a PLL. The phase detector compares the frequency of the VCXO and RXRECCLK and adjusts the VCXO so that its frequency matches that of RXRECCLK.

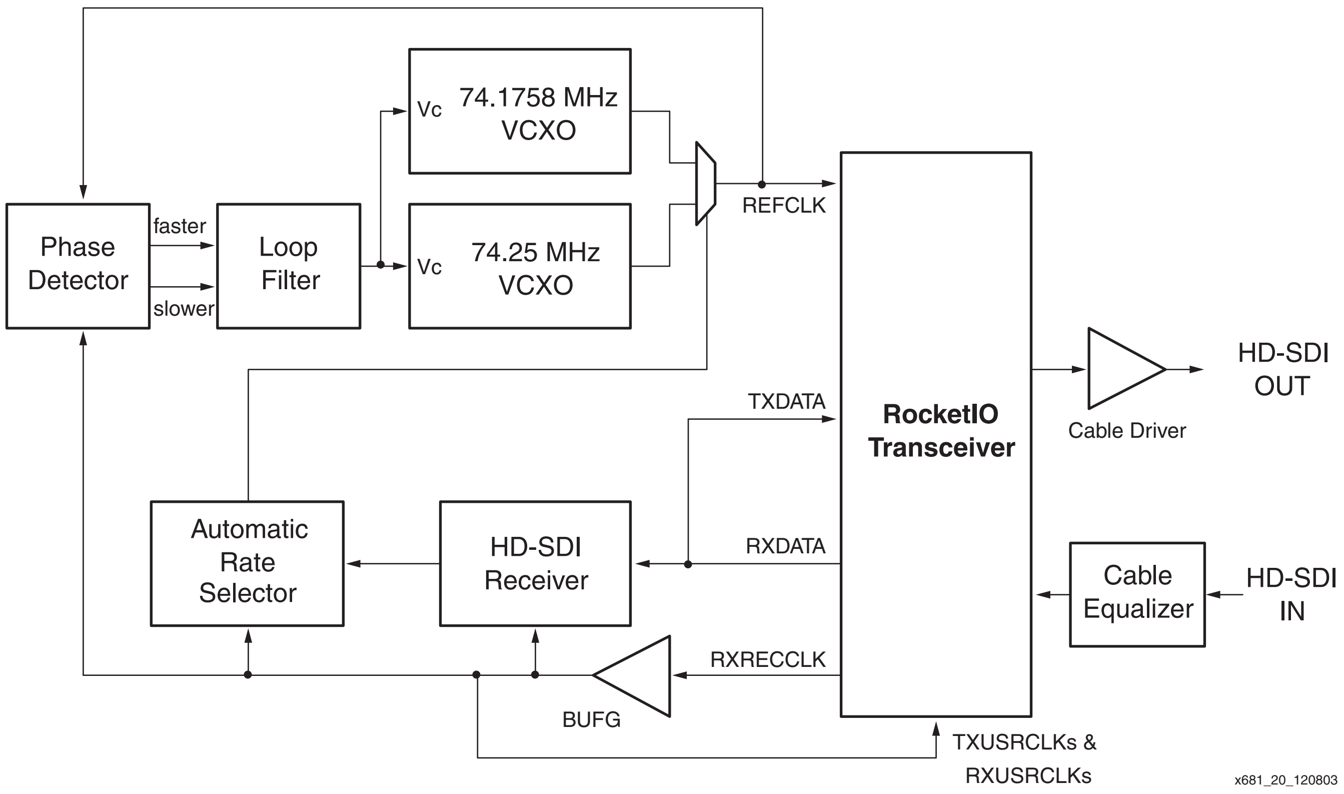


Figure 20: **Using a VCXO to Lock the Reference Clock to RXRECCLK**

This scheme solves the initialization problem of trying to spin up the CDR unit in the RocketIO receiver prior to having a stable RXRECCLK. Recall that during spin up, RXRECCLK is equal to the reference clock. If RXRECCLK is supplying the reference clock, then there is a cyclic problem with initializing the system. Because the *pull range* of a VCXO (the range over which the frequency of the VCXO can be adjusted by the control input) is limited to within something like ±100 ppm, the VCXO always is close in frequency to its normal center frequency, allowing the CDR unit to spin up and lock to the incoming bitstream frequency. During the initial stages of the CDR spin-up process, RXRECCLK is set equal to the reference clock input (the VCXO's output). The phase detector, seeing that RXRECCLK is the same frequency as the VCXO, does not try to adjust the frequency of the VCXO. As the CDR unit begins to lock to the bitstream, RXRECCLK starts to move towards the bitstream frequency, and the phase detector forces the VCXO to track this frequency change, keeping the reference clock locked to the frequency of RXRECCLK.

As shown in Figure 20, this technique usually requires two VCXOs, one for each of the two HD-SDI reference clock frequencies because the pull range of most VCXOs is not sufficient to allow one VCXO to operate at both HD-SDI frequencies.

Using a VCO instead of a VCXO is a tempting consideration because VCOs typically have a much larger pull range than a VCXO, possibly allowing one VCO to run at both HD-SDI bit rates. However, VCOs typically have more jitter than VCXOs, so a low-jitter VCO is required. Also, some sort of mechanism is required to force the VCO to run at the two HD-SDI frequencies of 74.25 MHz and 74.1758 MHz under control of a supervisor circuit to provide the necessary reference for CDR spin up. After spin up, the VCO is controlled by the phase detector and tracks the RXRECCLK frequency.

Appendix C: A Low-Cost Reference Clock Solution

As documented earlier in this application note, the RocketIO transceiver requires two reference clock frequencies of 74.25 MHz and 74.25 / 1.001 MHz in order to support the two HD-SDI bit rates. These two reference clock frequencies can be provided using either two crystal oscillators (one for each frequency) or a frequency synthesizer.

One possible low-cost frequency synthesizer is the ICS660 Digital Video Clock Source (http://www.icst.com). This device can synthesize a number of different video related frequencies from one input reference clock. The ICS660 can produce both 74.25 MHz and 74.25 / 1.001 MHz from either a 13.5 MHz or 27 MHz reference. It also can produce a 74.25 / 1.001 MHz from a 74.25 MHz reference clock and 74.25 MHz from a 74.25 / 1.001 MHz reference clock.

Currently, the jitter produced by the ICS660 makes it marginal for use as a reference clock source for the RocketIO transceiver. Testing of the ICS660 on the Xilinx SDV demo board shows that the ICS660 can be used as a reference clock source for implementing an HD-SDI receiver. But when used as a reference clock source for an HD-SDI transmitter, the output alignment jitter of the HD-SDI transmitter is at the maximum amount allowed by the SMPTE 292M specification. As of this writing, ICS is planning a new part based on the ICS660 that will improve its jitter characteristics with the intention of making it suitable for use as a RocketIO reference clock source for both an HD-SDI transmitter and an HD-SDI receiver. Until then, the ICS660 can be used as a reference clock source for HD-SDI receivers built with RocketIO transceivers, but not for HD-SDI transmitters.

In Figure 21, an ICS660 uses a 27 MHz crystal from which it can synthesize either 74.25 MHz or 74.25 / 1.001 MHz on its output. The output of the ICS660 is connected to an IOB of the Virtex-II Pro FPGA. Internally, this signal is connected directly to the reference clock input of the RocketIO transceiver. A single output from the Virtex-II Pro FPGA commands the ICS660 to generate either 74.25 MHz or 74.25 / 1.001 MHz. This output can come from a module like hdsdi_rx_autorate, which toggles this signal periodically until the RocketIO transceiver locks to the incoming bitstream.

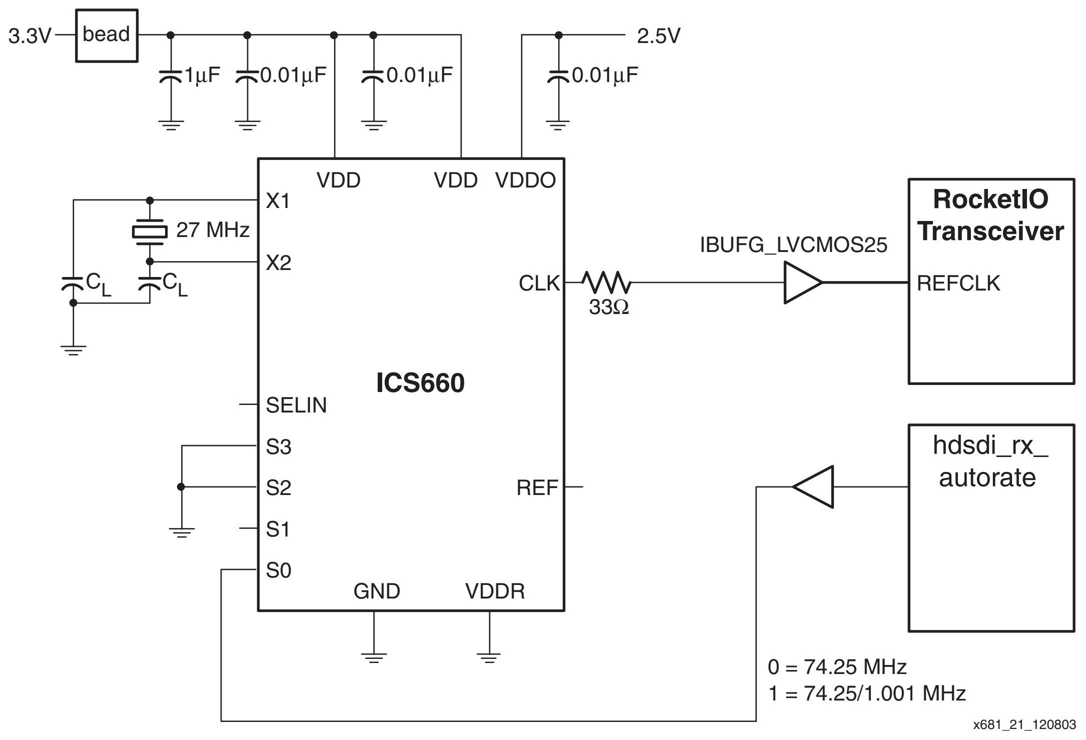


Figure 21: **ICS660 Providing REFCLK to RocketIO Transceiver**

In order to get the best performance from the ICS660, follow all the guidelines given in the ICS660 data sheet for power supply filtering and layout. Note that the REF clock output is disabled by grounding VDDR, which reduces the output jitter on the CLK output. Also note that the VDDO supply pin is connected to 2.5V to make the CLK output compatible with the 2.5V I/O standards of the Virtex-II Pro FPGA. The SELIN pin selects between using a crystal as the reference clock to the ICS660 (when High) or using an external clock source (when Low). SELIN has an internal pull-up resistor. The S[3:0] inputs also have internal pull up resistors. To generate 74.25 MHz and 74.25 / 1.001 MHz, the settings of S[3:0] are as follows:

- S3 and S2 are Low
- S1 is High
- S0 is Low to generate 74.25 MHz and High to generate 74.25 / 1.001 MHz

The S0 pin is driven with either the LVCMOS33 or the LVCMOS25 standard from the Virtex-II Pro FPGA.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/12/03	1.0	Initial Xilinx release.