XAPP1249 (v1.2) April 1, 2018

Implementing SMPTE SDI Interfaces with 7 Series GTX Transceivers

Author: SaiRam Nedunuri, Kalyanchakravathy Podalakuri

Summary

The Society of Motion Picture and Television Engineers (SMPTE) serial digital interface (SDI) family of standards is widely used in professional video equipment. These interfaces are used in broadcast studios and video production centers to carry uncompressed digital video, along with embedded ancillary data such as multiple audio channels. 6G-SDI and 12-SDI, collectively referred to as UHD-SDI, are recent extensions to the SDI family of standards that provide more bandwidth to transport Ultra HD video formats and higher frame rate HD video formats.

The Xilinx LogiCORE™ SMPTE UHD-SDI IP is a UHD-SDI receive/transmit datapath that does not have any device-specific control functions. This application note provides a module containing control logic to couple the UHD-SDI IP with the 7 series FPGA GTX transceivers to form a complete UHD-SDI interface. This application note also provides an example SDI design that runs on the KC705 board.

Introduction

The Xilinx® LogiCORE IP SMPTE UHD-SDI core (called the UHD-SDI core in the rest of this document) can be connected to a GTX transceiver in a Xilinx 7 series FPGA to implement an SDI interface capable of supporting the SMPTE SD-SDI, HD-SDI, 3G-SDI, 6G-SDI, and 12G-SDI standards. The UHD-SDI core and GTX transceiver must be supplemented with some additional logic to connect them together and implement a fully functional UHD-SDI interface. This application note describes this additional control and interface logic and provides necessary control and interface modules in Verilog source code.

In this application note, the term SDI is used to generically refer to the SMPTE family of interfaces standards including SD-SDI, HD-SDI, 3G-SDI, 6G-SDI and 12G-SDI.

7 series GTX transceivers can support all SDI bit rates up to and including 12G-SDI. Rates up to 6G-SDI can be supported with the GTX transceivers in -1 speed grade devices. However, 12G-SDI bit rates are only supported with the GTX transceivers in -3 speed grade devices and only in certain packages due to line rate limitations of the GTX transceivers. Refer to *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* [Ref 13] and *Virtex-7 T and XT FPGAs Data Sheet: DC and AC Switching Characteristics* [Ref 14] for the maximum line rates supported by the GTX transceivers for each combination of speed grade and device package.

The primary functions of the device-specific SDI control logic provided with this application note are:

- Reset logic for the GTX transceiver
- Dynamic switching of the GTX RX and TX serial clock dividers to support the five SDI standards
- Dynamic RX and TX reference clock switching to support two different bit rates in each of the HD-SDI, 3G-SDI, 6G-SDI, and 12G-SDI standards
  - 1.485 Gb/s and 1.485/1.001 Gb/s in HD-DI mode
  - 2.97 Gb/s and 2.97/1.001 Gb/s in 3G-SDI mode
  - 5.94 Gb/s and 5.94/1.001 Gb/s in 6G-SDI mode
  - 11.88 Gb/s and 11.88/1.001 Gb/s in 12G-SDI mode
- Dynamic switching of the GTX RXDATA and TXDATA port widths
  - 20-bit RXDATA and TXDATA ports for SD-SDI, HD-SDI, and 3G-SDI modes
  - 40-bit RXDATA and TXDATA ports for 6G-SDI and 12G-SDI modes
- Data recovery unit for recovering data in SD-SDI mode
- RX bit rate detection to determine if the RX is receiving integer frame-rate signals (line rates such as 1.485 Gb/s and 2.97 Gb/s) or fractional frame-rate signals (line rates such as 1.485/1.001 Gb/s and 2.97/1.001 Gbs)

Also supplied with this application note is a wrapper file that contains a GTX transceiver instance, an instance of the control module, and an instance of the SMPTE UHD-SDI core with the necessary connections between them. This file simplifies the process of creating an SDI interface.

This application note includes an example SDI design using the UHD-SDI core. This example design runs on the KC705 evaluation board. A Fidus 12G-SDI FPGA mezzanine card (FMC) is also required to provide the UHD-SDI physical interfaces.

In this document, the following terms are used. The UHD-SDI core refers to the SMPTE UHD-SDI core that is available in the Vivado® tool IP catalog starting with 2015.1 release. The control module is a module that implements the various device-specific functions when using the GTX to implement an UHD-SDI interface using the UHD-SDI core. The control module is supplied on source code form with this application note. The GTX wrapper is a wrapper file for a single GTX transceiver generated by the 7 Series FPGAs Transceivers Wizard that is available in the IP catalog. The GTX common wrapper is a wrapper file containing the QPLL for the GTX Quad also generated by the 7 Series FPGAs Transceivers Wizard when the GTX wrapper is generated. The SDI wrapper is a wrapper module that instances and interconnects the SMPTE UHD-SDI core, the GTX wrapper, and the control module. The SDI wrapper is supplied in source code form with this application note. The GTX common wrapper is not included in the SDI wrapper and must be instantiated separately in the application. Figure 1 is a simplified block diagram of how the various pieces fit together to form an UHD-SDI interface.

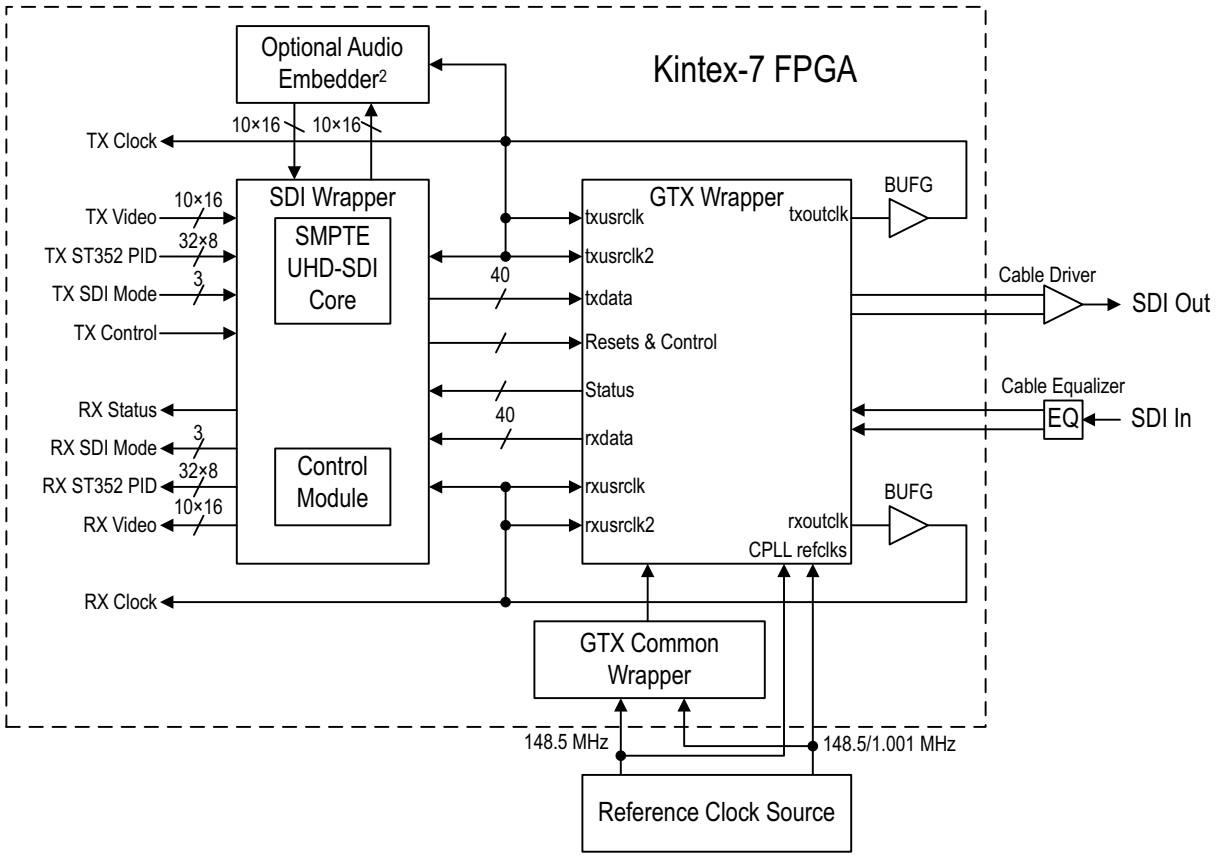

Figure 1: **Block Diagram of a Complete UHD-SDI RX/TX Interface**

Features

The *LogiCORE IP SMPTE UHD-SDI Product Guide* [Ref 16] lists all the features of the UHD-SDI core and SMPTE standards supported by the core. Please refer to that document for more information. That document also has timing diagrams showing the input and output timing of the core in the various SDI modes.

This document uses the term elementary data streams to refer to an SDI data stream that is not multiplexed. For example, an HD-SDI signal consists of two elementary data streams, usually referred to as the Y and C data streams, that are multiplexed together onto the virtual 10-bit HD-SDI interface. Likewise, a 3G-SDI level A signal also consists of two elementary data streams, called data stream 1 and data stream 2, that are multiplexed together onto the 10-bit virtual 3G-SDI interface. A 3G-SDI level B signal, however, consists of four elementary data streams, a Y and a C data stream for each of the HD-SDI signals that are aggregated together onto the 3G-SDI level B interface. These four elementary streams get interleaved in a 4-way multiplex onto the 10-bit virtual 3G-SDI interface. With the introduction of 6G-SDI and 12G-SDI, up to 16 elementary data streams may be interleaved onto a single SDI interface. These data streams are called ds1 through ds16 in this document and in the port names of the UHD-SDI core and the UHD-SDI wrapper.

The UHD-SDI core TX only accepts and the RX only outputs elementary, non-multiplexed, data streams on its data stream inputs and outputs. The multiplexing and de-multiplexing of data streams occurs internally to the UHD-SDI core and is not something that must be dealt with outside of the core. SD-SDI is an exception to this. The ST 259 SD-SDI standard defines a single data stream that carries both the Y and C components. This is considered to be an elementary data stream by the UHD-SDI core because multiple EAVs and SAVs are not interleaved.

The UHD-SDI core does not do any mapping between native video formats and elementary data streams. The user application must do any necessary mapping of video to elementary data streams prior to providing those streams to the UHD-SDI transmitter and must reconstruct the video image from the elementary streams output by the UHD-SDI receiver. For all video formats on SD-SDI and single-link HD-SDI and for 1080p 50, 59.94, and 60Hz 4:2:2 YCbCr 10-bit video on 3G-SDI level A, no mapping is necessary because there is a one-to-one correspondence between the data streams of these formats and the elementary data streams into and out of the UHD-SDI core. This is also true for 3G-SDI level B-DS, the dual stream mode where two HD-SDI video formats are aggregated onto a single 3G-SDI interface. For dual-link HD-SDI, 3G-SDI level B-DL, multi-link 3G-SDI, 6G-SDI, and 12G-SDI, mapping of the video formats to and from elementary data streams is required and is not done in the UHD-SDI core.

For 6G-SDI, the UHD-SDI core supports up to 8 elementary data streams. For 12G-SDI, the UHD-SDI core supports up to 16 elementary data streams. In the SMPTE 6G-SDI and 12G-SDI mapping documents, the term "data streams" refers to both multiplexed and non-multiplexed (elementary) data streams and care must be used when interpreting these documents to determine how many elementary data streams are used by each mapping method. Depending on the data format being transported, either four or eight elementary data streams are interleaved together on a 6G-SDI interface and either eight or sixteen elementary data streams are interleaved together on a 12G-SDI interface. The 16-way interleave only occurs in dual link 12G-SDI. The UHD-SDI TX must be told how many streams are active on its input using the port called tx_mux_pattern. The UHD-SDI RX automatically determines how many elementary data streams are present in the incoming SDI signal, demultiplexes the data stream appropriately, and indicates on the rx_active_streams port how many elementary data streams are present in the incoming signal.

Using 7 Series GTX Transceivers for SDI Interfaces

This section describes the details of implementation UHD-SDI interfaces using GTX transceivers in Xilinx 7 series FPGA devices (Kintex®-7, Virtex®-7, and Zynq™-7000).

The information in this section is intended to supplement, not replace, the information in *7 Series GTX/GTH Transceivers User Guide* [Ref 15]. This information highlights features of the GTX transceivers that are of particular importance for UHD-SDI applications.

There are several clocks required in applications using GTX transceivers. The SDI protocol does not allow for clock correction by adding and removing extra data in the data stream. Therefore, careful attention is required regarding how these clocks are generated and used in the application. GTX transceivers require reference clocks to operate. The reference clocks are used by phase-locked loops (PLLs) in the GTX transceiver Quad to generate serial clocks for the

receiver and transmitter sections of each transceiver. As described in more detail in the GTX Transceiver Reference Clocks section, the serial bit rate of the GTX transmitter is an integer multiple of the reference clock frequency it is using. Furthermore, the data rate of the video provided to the input of the UHD-SDI transmitter datapath must also exactly match (or be a specific multiple of) the frequency of the reference clock used by the GTX transmitter. Consequently, a UHD-SDI application must generate the transmitter reference clock so that it is frequency-locked exactly with the data rate of the video stream being transmitted.

The GTX transmitter outputs a clock on its txoutclk port at a frequency that is exactly equal to the word rate of the data that must enter the txdata port of the GTX transmitter. The txoutclk is generated in the GTX transmitter by dividing the serial clock from the PLL down to the word rate. In most applications, the txoutclk from the GTX transmitter is buffered by a global (BUFG) clock buffer and then used to clock the UHD-SDI transmitter datapath and the txusrclk and txusrclk2 clock inputs of the GTX transmitter. It is possible to use a clock other than one derived directly from txoutclk as the clock source for the UHD-SDI transmitter datapath and the txusrclk and txusrclk2 ports of the GTX transmitter. A shallow TX buffer in the GTX transmitter does allow for phase differences between the data entering the txdata port and the internal clock of the GTX transmitter. However, any frequency difference between the incoming data and the internal clock frequency of the GTX transmitter (as represented by txoutclk) quickly causes the TX buffer to underflow or overflow, resulting in errors in the serial bitstream generated by the GTX transmitter. Consequently, the data rate of the data stream entering the txdata port of the GTX transmitter (as represented by the frequency of the txusrclk and txusrclk2 clocks) and the internal data rate of the GTX transmitter (as set by the transmitter reference clock and represented by the frequency of txoutclk) must match exactly.

The GTX receiver reference clock, however, does not need an exact relationship with the line rate of the incoming SDI signal. This is because the clock and data recovery (CDR) unit in the GTX receiver can receive lines rates that are up to ±1250 ppm away from the nominal bit rate as set by the reference clock frequency in all modes except 12G-SDI. In 12G-SDI mode, the line rate of the 12G-SDI signal must be within ±200 ppm of the nominal line rate as set by the reference clock frequency. This allows the receiver reference clock to be generated by a local oscillator that has no exact frequency relationship to the incoming SDI signal. The GTX receiver generates a recovered clock that is frequency-locked to the incoming SDI bit rate. This clock is output on the rxoutclk port of the GTX transceiver. As is described in more detail later in this application note, rxoutclk is a true recovered clock when receiving any SDI signal except SD-SDI. Typically, rxoutclk is buffered by a global clock buffer and then applied to the rxusrclk and rxusrclk2 ports of the GTX receiver and used as the clock for the UHD-SDI receiver datapath.

One additional clock is required for SDI applications. This is a free-running, fixed-frequency clock that is used as the clock for the dynamic reconfiguration port (DRP) of the GTX transceiver. This same clock is also usually supplied to the control module in the SDI wrapper where it is used for timing purposes. Xilinx recommends that the frequency of this clock be at least 10 MHz. The maximum frequency of this clock is limited by the maximum allowed DRP clock frequency of the GTX transceiver, which is speed grade dependent. The frequency of this clock does not require any specific relationship relative to other clocks or data rates of the SDI application. This clock must not change frequencies when the SDI mode changes. It must remain running at the same nominal frequency at all times. It also must never stop while the SDI application is active. This clock can be used for all SDI interfaces in the device.

The frequency of the rxoutclk and txoutclk depend on the SDI mode and the width of the GTX transceiver's rxdata and txdata ports. This relationship is fixed by the architecture of the GTX transceiver. The RX and the TX both use clock enables to throttle the data stream transfer data rate because, in some cases, the data rate on the data streams is less than the frequency of the clock. Table 1 shows the relationships between SDI mode, number of active data streams, rxdata/txdata port widths, rxoutclk/txoutclk frequencies, and clock enable cadences. The clock enable cadences are given in number of clocks between assertions of the clock enable over two data word cycles where 1/1 means that the clock enable is asserted every clock cycle, 2/2 indicates assertion every other clock cycle (50% duty cycle), 4/4 indicates assertion every fourth clock cycle (25% duty cycle), and 5/6 indicates that the clock enable alternates between assertion every 5 or 6 clock cycles, to average once every 5.5 clock cycles (one instance of 5 clock cycles between High pulses on the clock enabled followed by one instance of 6 clock cycles between High pulses on the clock enable, with this pattern repeating).

Table 1: **Clock Frequencies and Clock Enable Requirements**

SDI-Mode	Active Data Streams	RX/TXDATA Bit Width	RX/TXOUTCLK Frequency	Clock enable
SD-SDI	1	20	148.5 MHz	5/6
HD-SDI	2	20	74.25 or 74.25/1.001 MHz	1/1
3G-SDI A	2	20	148.5 or 148.5/1.001 MHz	1/1
3G-SDI B	4	20	148.5 or 148.5/1.001 MHz	2/2
6G-SDI	4	40	148.5 or 148.5/1.001 MHz	1/1
6G-SDI	8	40	148.5 or 148.5/1.001 MHz	2/2
12G-SDI	8	40	297 or 297/1.001 MHz	2/2
12G-SDI	16	40	297 or 297/1.001 MHz	4/4

GTX Transceiver Reference Clocks and PLLs

7 Series GTX transceivers are grouped into quads. Each Quad contains four GTXE2_CHANNEL transceiver primitives and one GTXE2_COMMON primitive containing a Quad PLL (QPLL) as shown in Figure 2. The clock generated by the QPLL is distributed to all four transceivers in the Quad. Each GTXE2_CHANNEL has its own PLL called the channel PLL (CPLL), which can provide a clock to the RX and TX of that transceiver only. Each RX and TX unit in the Quad can be individually configured to use either the QPLL or the CPLL as its clock source. Furthermore, any RX or TX unit can dynamically switch its clock source between the QPLL and the CPLL. This configuration and the dynamic switching capability are particularly useful for SDI applications.

IMPORTANT: *The CPLL and QPLL have maximum line rates of 6.6 Gbps and 12.5 Gbps, respectively. This means that CPLL can only be used up to 6G-SDI line rate while the QPLL can support up to 12G-SDI. It is important to note that only -3 speed grade 7 Series GTX transceiver, QPLL has a maximum line rate of 12.5 Gbps thus the only speed grade that can support up to 12G-SDI. See GTX Transceiver Switching Characteristics in the Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS182) [Ref 13] for details.*

Typical UHD-SDI applications require the GTX transceivers to support up to nine different bit rates:

- 270 Mb/s for SD-SDI
- 1.485 Gb/s for HD-SDI
- 1.485/1.001 Gb/s for HD-SDI
- 2.97 Gb/s for 3G-SDI
- 2.97/1.001 Gb/s for 3G-SDI
- 5.94 Gb/s for 6G-SDI
- 5.94/1.001 Gb/s for 6G-SDI
- 11.88 Gb/s for 12G-SDI
- 11.88/1.001 Gb/s for 12G-SDI

The clock and data recovery (CDR) unit in the RX section of the GTX transceiver can support receiving bit rates that are up to +/-1250 ppm from the reference frequency at bit rates less than 6.6 Gb/s. HD-SDI, 3G-SDI, 6G-SDI, and 12G-SDI each have two bit rates that differ by exactly 1000 ppm. For HD-SDI, 3G-SDI, and 6G-SDI, both bit rates can be received using a single reference clock frequency. That same reference clock frequency can also support reception of SD-SDI. Thus, for all SDI modes except 12G-SDI, just a single RX reference clock frequency is required. However, at 12G-SDI rates, the CDR unit has only ±200ppm tolerance relative to the reference clock frequency. Thus two different reference clock frequencies are needed to receive the two 12G-SDI bit rates. These two reference clock frequencies are typically 148.5 MHz to receive 11.88 Gb/s and 148.5/1.001 MHz to receive 11.88/1.001 Gb/s.

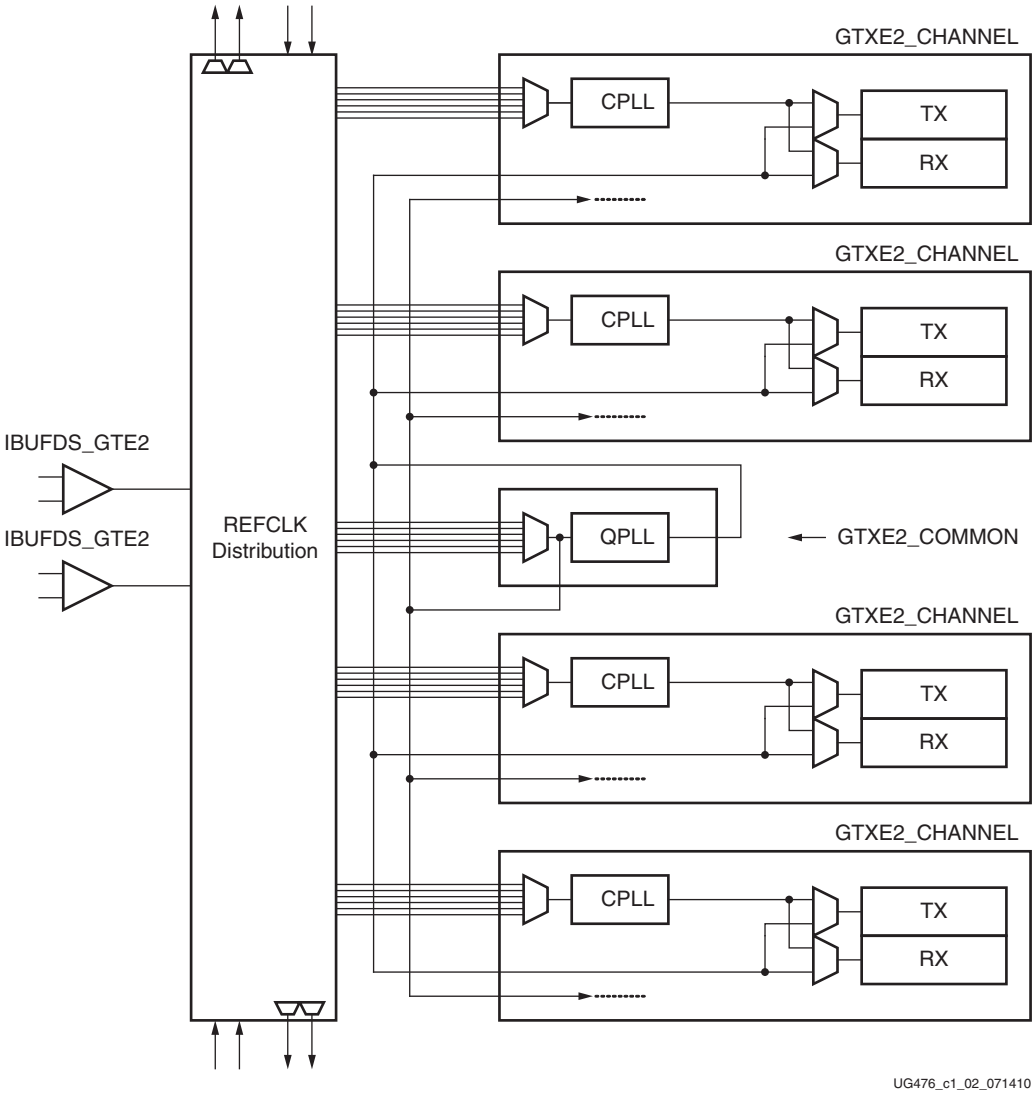

Figure 2: **7 Series GTX Quad Configuration**

The TX section of the GTX transceiver requires two different reference frequencies to support all the SDI bit rates. This is because the transmitters, in general, can only transmit at an exact integer multiple of the supplied reference clock frequency. Some SDI applications may be configured only to support fractional frame rates where the HD-SDI, 3G-SDI, 6G-SDI, and 12G-SDI bit rates are always the X/1.001 bit rates. These bit rates can all be generated using a reference clock of 148.5/1.001 MHz. However, transmitting SD-SDI still requires a reference clock of 148.5 MHz because it always has a bit rate of exactly 270 Mb/s and never 270/1.001 Mb/s.

Therefore, most SDI applications provide two separate reference clocks to the GTX Quad. Usually, the supplied reference frequency pair are 148.5 MHz and 148.5/1.001 MHz. This application note always refers to the reference clock frequency pair 148.5 MHz and 148.5/1.001 MHz.

The source of the GTX transceiver reference clocks for SDI applications is very application-specific. The receiver reference clock source can be a local oscillator because it does not need to match the incoming SDI bit rate exactly. However, because the GTX transmitter line rate is always an integer multiple of the reference clock frequency, the frequency of the

transmitter reference clock must be exactly related to the data rate of the transmitted data. Most often, the transmitter reference clocks are generated by genlock PLLs, thereby deriving the GTX transmitter line rate from the studio video reference signal. In some cases, such as the SDI pass-through demonstration included with this application note, the transmitter line rate is derived from the recovered clock of the GTX receiver that is receiving the SDI signal. In such cases, an external PLL is required to reduce the jitter on the recovered clock before using it as the transmitter reference clock.

PLL Configuration at 6G-SDI and Slower

If an application is only supporting 6G-SDI and lower, and not 12G-SDI, then the typical use case is to supply one reference clock to the QPLL and use the QPLL to clock all of the GTX receivers in the Quad. The receivers are able to receive all of the rates from 6G-SDI and slower with that single reference clock frequency. The second reference clock frequency is distributed to all the CPLLs in the Quad. The GTX transmitters are dynamically switched between the QPLL and the CPLL using the TXSYSCLKSEL port. This configuration is shown in Figure 3.

When 6G-SDI is the maximum rate supported, the QPLL is operated in range 1. This is important because the 7 series GTX transceivers have only been characterized to receive 3G-SDI and HD-SDI with the QPLL as the clock source when the QPLL is operating in range 1.

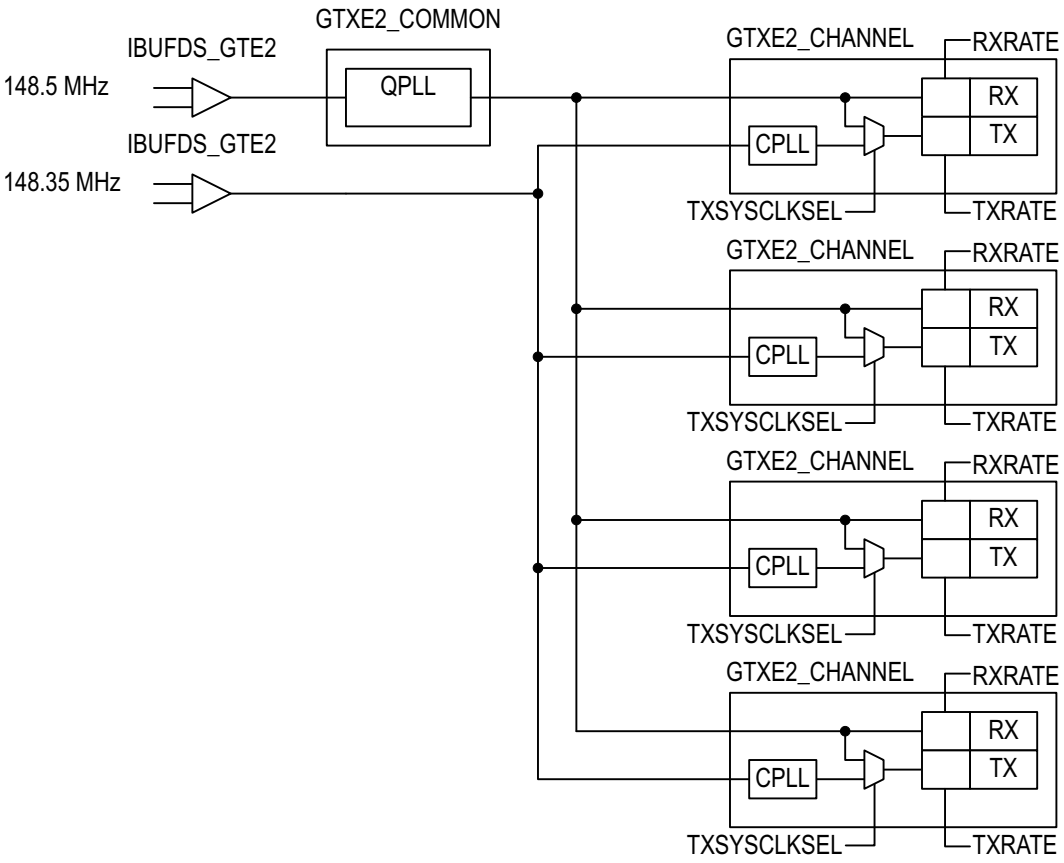

Figure 3: **Typical PLL Configuration for Applications Not Supporting 12G-SDI**

PLL Configuration for 12G-SDI Operation

12G-SDI support on 7 series GTX transceivers imposes additional requirements on clocking and limitations on operation of the transceivers. These requirements and limitations are:

- Only the QPLL can be used as the clock source for any RX and TX operating at 12G-SDI and the QPLL must be operating in range 2.
- The 7 series GTX RX has not been characterized at 3G-SDI and HD-SDI rates with the QPLL operating in range 2. When the QPLL is operating in range 2, any GTX RX must use the CPLL as its clock source for 3G-SDI and HD-SDI. TX units may use the QPLL to transmit 3G, HD, or SD even if it is operating in range 2.
- At 12G-SDI line rates, the GTX CDR only has ±200ppm tolerance. Thus, the QPLL must have a 148.5 MHz reference clock when the transceivers in the Quad are running at the 11.88 Gb/s line rate and it must have a 148.5/1.001 MHz reference clock when the transceivers in the Quad are running at 11.88/1.001 Gb/s line rate.
- Because there is only a single QPLL available in the Quad, it is not possible to support both 11.88 Gb/s and 11.88/1.001 Gb/s 12G-SDI line rates simultaneously in the same Quad.

There are several strategies for working with the 12G-SDI clocking restrictions. If these restrictions are too cumbersome for a particular application, consider using the UltraScale Kintex device. The UltraScale Kintex devices has two QPLLs per GTH Quad, making it possible to support mixed operation at both 12G-SDI rates simultaneously in the same Quad.

Some applications may only need to support one 12G-SDI line rate at a time and don't dynamically change between the two 12G-SDI rates. In other words, in 12G-SDI mode, all transceivers in the Quad only operate at either 11.88 Gb/s or 11.88/1.001 Gb/s and never switch back and forth between these two line rates. In this use case, all other SDI line rates can be supported, including any mix of integer and fractional frame rates at 6G-SDI and slower.

In this use case, shown in Figure 4, the QPLL is given a single reference clock frequency, which can be either 148.5 MHz or 148.5/1.001 MHz, depending on which 12G-SDI line rate is to be supported (148.5 MHz for 11.88 Gb/s or 148.5/1.001 MHz for 11.88/1.001 Gb/s). In the example shown in the figure, the Quad only supports 11.88 Gb/s, so the QPLL reference clock frequency is 148.5 MHz. The QPLL operates in range 2 at 11.88 GHz and provides a 5.94 GHz clock to each transceiver in the Quad. (The transceivers always use a clock from the PLLs that is exactly one half the line rate).

The CPLLs are all given the other reference clock frequency, the one not given to the QPLL. In this example, the CPLLs are given the 148.5/1.001 MHz reference clock and operate at 2.97/1.001 GHz, providing a clock to each transceiver.

Any RX or TX in the Quad running at 11.88 Gb/s must use the QPLL clock as its serial clock source and must have its PLL divider set to divide by 1. At 6G-SDI rates, the RX can use either the QPLL or the CPLL as long as the correct divider value is used (divide by 1 when using the CPLL and divide by 2 when using the QPLL). At 3G-SDI rates and lower, the RX must use the CPLL because the QPLL is in range 2. The TX units use the QPLL when transmitting integer frame rate SDI line rates and the CPLL when transmitting non-integer frame rate SDI lines rates. In this scenario, the only limitation is that only the 11.88 Gb/s 12G-SDI line rate is supported. It is not

possible to transmit or receive at 11.88/1.001 Gb/s given the arrangement of the reference clocks.

If the QPLL is given the 148.5/1.001 MHz reference clock and the CPLL is given the 148.5 MHz reference clock, then this use case supports the 11.88/1.001 Gb/s line rate, but not the 11.88 Gb/s line rate. All slower line rates can be supported.

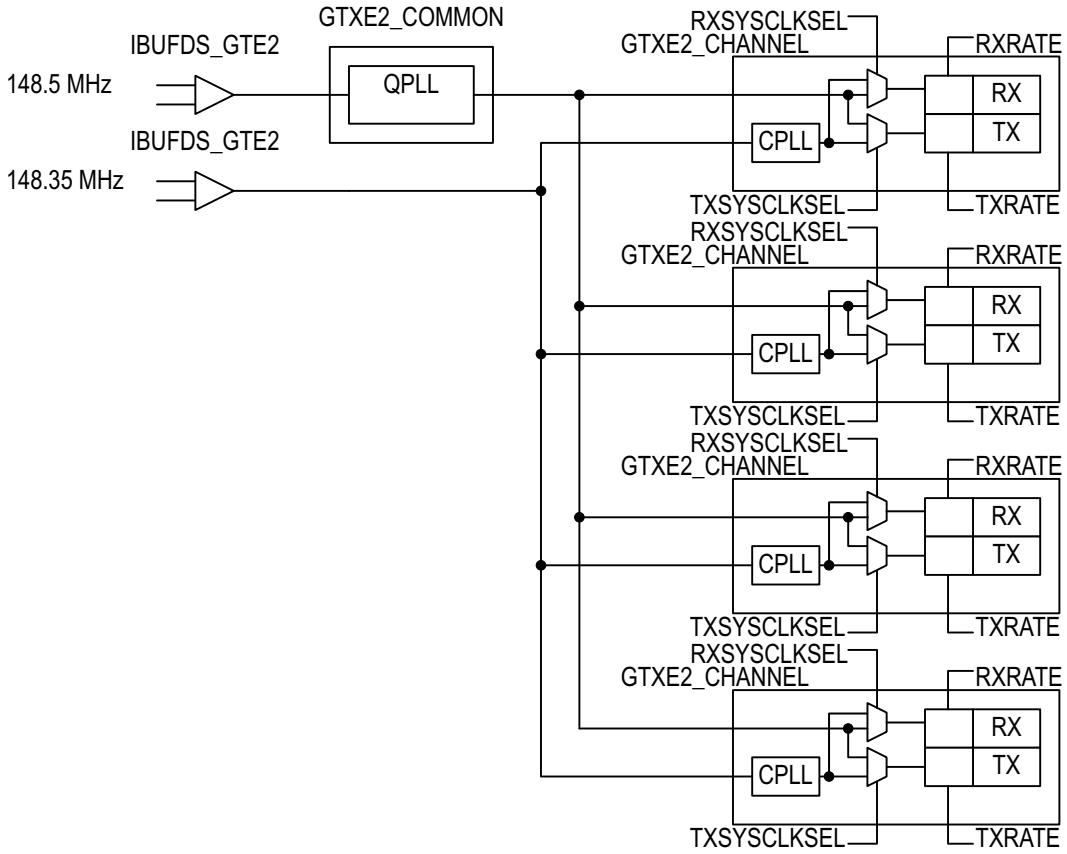

Figure 4: **PLL Configuration for Applications Supporting a Single 12G-SDI Line Rate**

If dynamic switching between the two 12G-SDI line rates is a requirement, then things get much more complicated. Dynamic switching is possible, but has significant implications.

To support dynamic switching between the two 12G-SDI line rates, the reference clock to the QPLL must be dynamically switched between 148.5 MHz and 148.5/1.001 MHz. Anytime the reference clock frequency to the QPLL is dynamically switched, the QPLL must be reset. The single 12G-SDI line rate supported by the entire Quad at any point in time is dictated by which reference clock frequency is given to the QPLL. Thus, it is possible to switch the entire Quad between 11.88 Gb/s and 11.88/1.001 Gb/s, but all transceivers in the Quad running in 12G-SDI mode always run at the 12G-SDI line rate dictated by the frequency of the QPLL reference clock. It is not possible to have some units in the Quad running at 11.88 Gb/s and others at 11.88/1.001 Gb/s. Of course, it is possible to have different GTX Quads running at different 12G-SDI line rates, but all transceivers in one Quad must all run at the same line rate when running in 12G-SDI mode.

Any RX unit that is using the QPLL as the serial clock source for 6G-SDI becomes upset when the reference clock to the QPLL is dynamically switched between reference clock frequencies and reset. Any TX unit in the Quad that is using the QPLL as the serial clock source not only experiences an upset when the QPLL is reset, but also experiences a 1000ppm shift in line rate as a result of the QPLL changing reference clock frequencies.

Thus, for most applications, support for dynamic switching between the two 12G-SDI line rates is problematic. The application can never support operation at both 12G-SDI line rates in the same GTX Quad simultaneously. And, any switch between 12G-SDI line rates impacts all transceivers in the Quad using the QPLL at the time of the switch.

There are several possible use cases where dynamic switching between 12G-SDI line rates may be practical. One such use case is shown in Figure 5. In this use case, each transceiver is used in one direction only, either as a receiver or as a transmitter. The top two transceivers in the figure are RX-only and the bottom two transceivers are TX-only. When operating at 6G-SDI line rates and slower, each RX or TX always uses its CPLL. The CPLL is dynamically switched between the two reference clocks as required using the CPLLREFCLKSEL port. Any RX or TX unit running at 12G-SDI rates, must use the QPLL as the clock source. The QPLL can be dynamically switched between the two reference clock frequencies as required. But all units operating in 12G-SDI mode at that time switch between the two 12G-SDI line rates simultaneously as the QPLL dynamically switches between reference clock frequencies. Any mix of RX and TX units in the Quad could be supported this way; it doesn't have to be two RX and two TX.

The reason for limiting each transceiver to just RX or just TX and not both is to make it easier to use the CPLL. Because the CPLL is needed for both RX and TX, sharing the CPLL is somewhat difficult. Anytime the CPLL is dynamically switched between reference clock sources to change the TX line rate, it would temporarily disrupt the RX if they were both active and using the clock from the CPLL. If, however, the application doesn't care that both the RX and TX are affected by dynamically switching the CPLL between the two reference clock frequencies, then it would be possible to use a transceiver to transmit and receive at the same time.

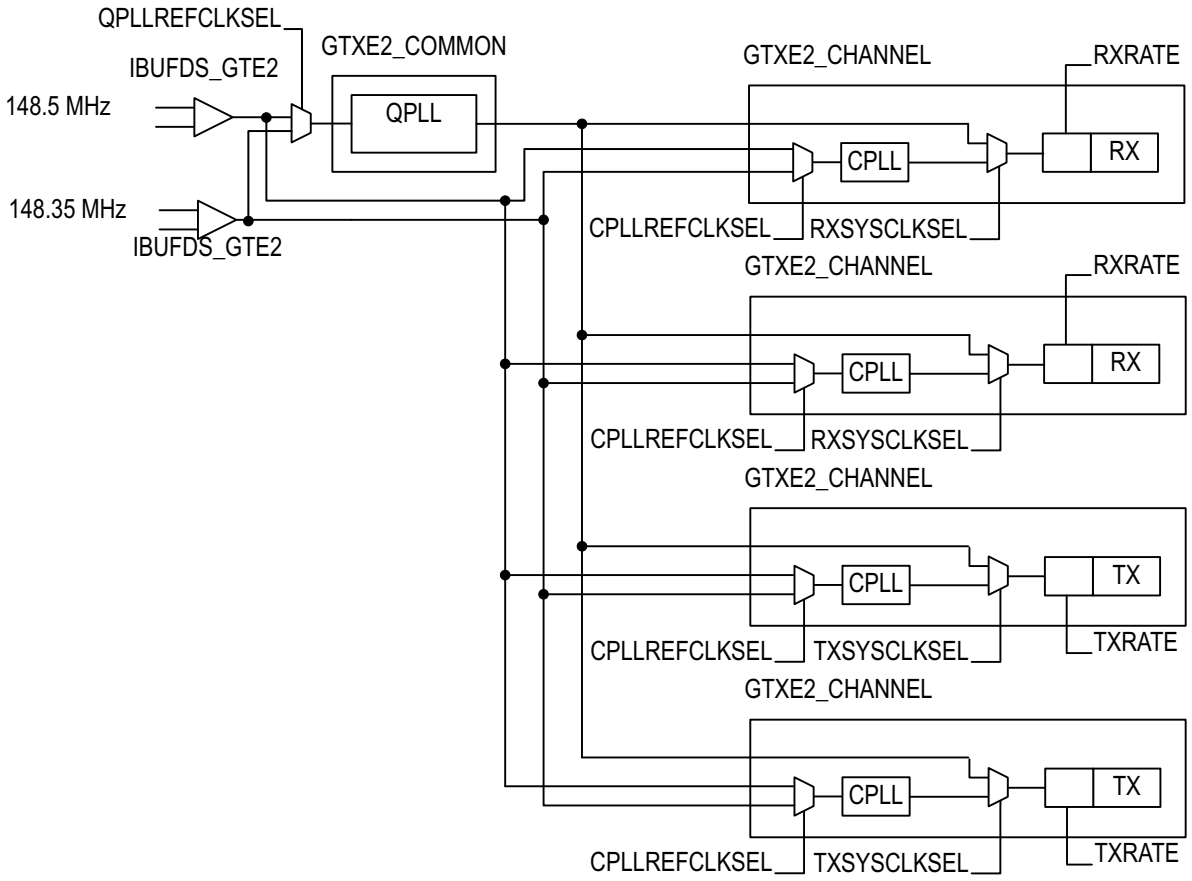

Figure 5: **Example PLL Configuration when Supporting Dynamic Switching Between 12G-SDI Rates**

Resets

The GTX transceiver has very specific reset requirements as described in the *7 Series GTX/GTX Transceivers User Guide* [Ref 15]. The GTX transceiver requires careful coordination of PLL resets, GTX transceiver resets (gttxreset and gtrxreset), dynamic changes of some GTX transceiver ports including the width of the txdata and rxdata ports, and dynamic changes of GTX transceiver attributes through the DRP. Without proper coordination of all of these events, it is possible for the GTX to fall into a state in which it does not function properly, a situation from which the only possible recovery is to reconfigure the FPGA. The control module supplied with this application note enforces all of these requirements to ensure proper operation of the GTX transceiver.

The user application should never directly control the GTX inputs gttxreset and gtrxreset. To ensure proper operation of the GTX transceiver, these GTX transceiver inputs must only be controlled by the SDI control module. The user application can request GTX resets using the various reset inputs of the control module. These reset requests are serviced at the next appropriate time by the control module, coordinating the resets with other GTX transceiver actions so that they do not interfere.

PLL Configuration and Control

The GT wizard generates a GTX Common wrapper when the GTX wrapper example design is generated. That GT wizard configures the QPLL in the GTX common module according to the parameters entered into the GTX wizard. If the maximum line rate supported is 6G-SDI, the QPLL is configured for operation in range 1. If the maximum line rate supported is 12G-SDI, the QPLL is configured for operation in range 2. This is important to understand because the UHD-SDI wrapper must be told which range the QPLL is configured for to properly control the GTX.

The UHD-SDI wrapper has several ports for both the RX and the TX which are related to the configuration and selection of the PLLs. For the TX section, these ports are:

- tx_pll_select_in: This port selects which PLL is to be used as the serial clock source for the GTX TX. This port can be changed dynamically to change the TX clock source. Whenever, tx_pll_select_in changes, the UHD-SDI wrapper changes the TXSYSCLKSEL port of the GTX to change the selected PLL. And, the UHD-SDI wrapper also resets the GTX TX. The PLL selected by the value on tx_pll_select_in is not fixed, but is controlled by the tx_pll_type_in port.
- tx_pll_type_in: Bit 0 of this port specifies which PLL is used when tx_pll_select_in is Low. Bit 1 of this port specifies which PLL is used when tx_pll_select_in is High. The CPLL is selected if the tx_pll_type_in bit is Low and the QPLL is selected if the tx_pll_type_in bit is High. So, for example, consider the case where the application uses the CPLL when tx_pll_select_in is Low and the QPLL when tx_pll_select_in is High. The application hardwires a value of 2'b10 on the tx_pll_type_in port specifying use of the CPLL when tx_pll_select_in is Low and the QPLL when tx_pll_select_in is High. The tx_pll_type_in port is usually statically configured by hardwiring the bits of the port. However, the SDI control logic does support and respond to dynamic changes on the tx_pll_type_in port.
- tx_pll_range_in: Bit 0 of this port specifies the operating range of the TX PLL (0 = range 1, 1 = range 2) when tx_pll_select_in is Low. Bit 1 of this port specifies the operating range of the TX PLL when tx_pll_select_in is High. The CPLL is always considered to be in range 1. As an example, consider the case where the QPLL is used in range 2 and is selected when tx_pll_select_in is High and the CPLL is selected when tx_pll_select_in is Low. A value of 2'b10 must be applied to the tx_pll_range_in port of the UHD-SDI wrapper. Bit 0 is Low because the CPLL is selected when tx_pll_select_in is Low and the CPLL is always considered to be in range 1. Bit 1 is High because the QPLL is selected when tx_pll_select_in is High and, in this example, the QPLL is configured for range 2.

The UHD-SDI wrapper has exactly the same set of ports as just described for the RX section. These ports are named rx_pll_select_in, rx_pll_type_in, and rx_pll_range_in. They behave the same way as the TX ports, but operate on the RX section instead of the TX section by controlling the RXSYSCLKSEL port of the GTX and the RX PLL divider.

The UHD-SDI wrapper has two PLL reset outputs called rx_pll_reset_out and tx_pll_reset_out. These are generated by the SDI control logic to reset the QPLL and the CPLL. These reset outputs are asserted automatically during the power-on initialization sequence of the GTX executed by the SDI control logic and also during full GTX TX and RX reset sequences initiated by the assertion of the tx_gtx_full_reset_in and rx_gtx_full_reset_in ports of the UHD-SDI

wrapper. It is up to the application to properly connect the rx_pll_reset_out and the tx_pll_rest_out ports to the proper PLL reset inputs based on the configuration and requirements of the application.

The UHD-SDI wrapper has a cpllreset_in input which should be driven by either the rx_pll_reset_out or the tx_pll_reset_out or some logical combination of those two to reset the CPLL. The GTX common wrapper has a qpllreset_in port which should be driven by a rx_pll_reset_out or tx_pll_reset_out (or combination thereof) of one or more UHD-SDI wrappers in the same Quad as the GTX common. Usually one UHD-SDI wrapper per Quad is designated as the QPLL master and is responsible for resetting the QPLL in that Quad.

As an example, all four transceivers in a GTX Quad are used for SDI. The RX units only use the QPLL and the TX units are dynamically switched between QPLL and CPLL (the configuration shown in Figure 3). In this configuration, the following reset connections could be used. The tx_pll_reset_out port of each UHD-SDI wrapper is connected to the cpllreset_in port of that same UHD-SDI wrapper. The rx_pll_reset_out port of one of the UHD-SDI wrappers is connected to the qpllreset_in port of the GTX common and that UHD-SDI wrapper is considered to be the QPLL master and is responsible for resetting the QPLL. The qplllock_out port of the GTX common must be connected to the qplllock_in port of every UHD-SDI wrapper in that Quad. The cplllock_out port of the UHD-SDI wrapper is a status indicator only and doesn't need to be connected to anything for proper operation.

GTX Transceiver Initialization Sequence

Immediately following FPGA configuration, the SDI control module performs initialization sequences for the QPLL, the CPLLs, and the RX and TX units of the GTX transceiver. The control module has separate controllers for the RX and the TX. These controllers perform the following initialization sequence. The sequence described here is for the RX. The TX initialization sequence is identical except that it uses TX ports rather than the RX ports described.

1. After waiting at least 500 ns following completion of FPGA configuration, assert the PLL reset and gtrxreset signals.
2. Wait until the rx_refclk_stable input is asserted, then negate the PLL reset.
3. Wait until the PLL lock signal is asserted, then negate the gtrxreset signal.
4. Wait until the rxresetdone signal is asserted, then indicate that the initialization sequence is complete.

Also, the GTX txuserrdy and rxuserrdy inputs must be properly controlled. The SDI wrapper generates both of these signals. It asserts txuserrdy after gttxreset is negated. Likewise, it asserts rxuserrdy after gtrxreset is negated. In step 2, step 3, and step 4 of the initialization sequence where the sequence is waiting on a condition to be satisfied, a timeout counter is running. If the timeout counter expires before the wait condition is satisfied, the state machine moves to a timeout state where it increments a retry counter and then cycles back in the initialization sequence and resumes the sequence. If the retry counter reaches its maximum count due to numerous timeouts, the initialization sequence fails and the state machine moves to a fail state, indicating failure of the initialization sequence.

PLL Resets

Besides being reset during the initialization sequences the SDI control module runs after FPGA configuration, a QPLL or CPLL must also be reset whenever there is a change in frequency or interruption of the reference clock supplied to that PLL. The reset is required to force the PLL to relock to the reference clock. The qpllreset input of the GTX common wrapper and the cpllreset_in input of the UHD-SDI wrapper are controlled by the SDI control module to implement the PLL resets. The user application should not assert the PLL resets directly. The SDI control module should always control the PLL resets. However, it is up to the user application to determine when PLL resets are required. When a PLL must be reset, the application must request that the SDI control module reset the PLL and all of the GTX RX and/or TX units using the serial clock from that PLL. The UHD-SDI wrapper has a rx_pllreset_out output and a tx_pllreset_out output. These are used to control the qpllreset input of the GTX common wrapper and the cpllreset_in input of the UHD-SDI wrapper. If a PLL is used by only one RX or one TX unit, it is a simple matter to connect the correct rx_pllreset or tx_pllreset output of the UHD-SDI wrapper to the corresponding PLL reset input port. But, when a PLL provides the serial clock to multiple RX and/or TX units, it is more complicated and careful consideration must be given to how the PLL resets are connected and controlled.

The UHD-SDI wrapper has inputs that the application should use to request a full reset of the GTX RX (rx_gtx_full_reset_in) and the GTX TX (tx_gtx_full_reset_in). Asserting either of these inputs causes the appropriate controller in the control module to execute the full initialization sequence of the RX or TX section of the GTX, including resetting the associated PLL. The user application must properly control the rx_gtx_full_reset_in and tx_gtx_full_reset_in inputs so that these initialization sequences are done whenever there is an interruption or change in the reference clock used by the PLL.

It is up to the user application to properly control the rx_refclk_stable_in and tx_refclk_stable_in inputs to the control module. These must be asserted only when the reference clocks to the PLLs are stable. As previously described, the initialization sequences wait until these inputs are asserted before negating the PLL resets. Driving the rx_refclk_stable_in or tx_refclk_stable_in inputs Low does not initiate a reset of the associated PLL. PLL resets are only initiated by asserting the rx_gtx_full_reset_in and tx_gtx_full_reset_in inputs to the control module. The rx_refclk_stable_in and tx_refclk_stable_in are only used to delay completion of the reset sequence after the reset sequence has been started by assertion of rx_gtx_full_reset_in or tx_gtx_full_reset_in.

GTX TX Resets

There are three conditions that require the TX portion of the GTX transceiver to be reset:

- Whenever the PLL that supplies the serial clock to the GTX TX is reset, the gttxreset port must be used to reset the TX section. This is done automatically after FPGA configuration by the SDI control module and whenever the user application asserts the tx_gtx_full_reset_in to the SDI wrapper, causing both the PLL and the GTX TX to be reset.
- The SDI control logic automatically resets the GTX TX using the gttxreset input whenever txsysclksel port is changed dynamically. The txsysclksel port is used to select between the QPLL and the CPLL as the serial clock source for the GTX TX. Each GTX transceiver has its own txsysclksel port and can independently switch its serial clock source between the two

PLLs. The txsyclksel port should not be controlled directly by the application. The SDI control module dynamically changes the txsysclksel port of the GTX transceiver in response to changes on its tx_pll_select_in input. When the control module detects a change on its tx_pll_select_in input, it first asserts the gttxreset signal, then changes txsysclksel, and then negates gttxreset. The sequence is complete after the GTX transceiver asserts its txresetdone output. At that point, the SDI control module indicates completion of the txsysclksel change by asserting its tx_change_done_out output.

- The SDI control logic automatically resets the GTX TX using the gttxreset port whenever the PLL divider in the GTX TX is changed by the SDI control logic through the DRP in response to a change on the tx_mode_in input port. As with changes of the txsysclksel port, the SDI control logic indicates completion of this reset by the tx_change_done_out output.

The UHD-SDI wrapper has three reset inputs for the TX section:

- tx_rst_in: When asserted High, this input resets the SDI TX datapath in the UHD-SDI core. It does not reset the GTX at all.
- tx_gtx_full_rest_in: When asserted High, this input resets both the PLL associated with the TX and then the TX section of the GTX transceiver (gttxreset). These two resets are sequenced so that the gttxreset does not complete until after the PLL reset is complete and the PLL is locked to its reference clock.
- tx_gtx_reset_in: When asserted High, this input resets only the TX section of the GTX transceiver (gttxreset). If the PLL is not locked when the gttxreset sequence begins, the gttxrest sequence does not complete until after the PLL is locked.

GTX RX Resets

As with the TX section, the user application should rely on the SDI control module to carefully coordinate all of the RX reset and dynamic change activities described here to prevent them from interfering with each other.

These conditions require rests of the GTX RX section:

- Whenever the PLL that supplies the serial clock to the GTX RX is reset, the gtrxreset port must be used to reset the RX section. This is done automatically after FPGA configuration by the SDI control module and whenever the user application asserts the rx_gtx_full_reset_in to the SDI wrapper, causing both the PLL and the GTX RX to be reset.
- Changes in the SDI mode between SD, HD, 3G, 6G, and 12G-SDI require changes various input ports of the GTX and also to various attribute settings through the DRP. The SDI control logic makes these changes automatically whenever the RX SDI mode changes. After the SDI control logic has made all of the required modifications, it resets the GTX RX section using the gtrxrest port of the GTX.

The UHD-SDI wrapper has three reset inputs for the RX section:

- rx_rst_in: When asserted High, this input resets the SDI RX datapath in the UHD-SDI core. It does not reset the GTX at all.

- rx_gtx_full_rest_in: When asserted High, this input resets both the PLL associated with the RX and then the RX section of the GTX transceiver (gtrxreset). These two resets are sequenced so that the gtrxreset does not complete until after the PLL reset is complete and the PLL is locked to its reference clock.
- rx_gtx_reset_in: When asserted High, this input resets only the RX section of the GTX transceiver (gtrxreset). If the PLL is not locked when the gtrxreset sequence begins, the gtrxreset sequence does not complete until after the PLL is locked.

SDI Electrical Interface

External SDI cable equalizers and cable drivers are required to convert the serial signals into and out of the GTX transceivers to SDI electrical standards.

An external SDI cable equalizer must be used to convert the single-ended 75Ω SDI signal to a 50Ω differential signal compatible with the receiver input signal requirements of the GTX transceiver. Appropriate SDI cable equalizers are available from several manufacturers. The differential outputs of these cable equalizers usually must be AC-coupled to the GTX receiver input signals. An example of interfacing a typical SDI cable equalizer to a GTX receiver is shown in Figure 6. 12G-SDI cable equalizers typically have built in reclockers, but that doesn't change any of the requirements for electrical interfacing to the GTX.

IMPORTANT: *The capacitance values of the AC coupling capacitors between the outputs of the external SDI cable equalizer and the serial inputs of the GTX RX must be large enough to pass the SDI pathological signals without significant signal droop. AC coupling capacitors with values of at least 1.0 µF are required and 4.7 µF capacitors are recommended. Some new generation SDI cable equalizers default to 600 mV differential swing levels on their outputs instead of the traditional 800 mV differential swing. When using an equalizer with 600 mV differential swing, 4.7 µF capacitors may not be sufficient to prevent excessive signal droop at SD-SDI rates. It is recommended that cable equalizers be configured with 800 mV differential swing.*

The differential inputs of the GTX RX have built-in differential termination. As described in *7 Series GTX/GTX Transceivers User Guide* [Ref 15], RX Termination Use Mode 3 is the recommended termination mode for the GTX RX inputs in SDI applications. The GTX internal programmable termination voltage should be set to 800 mV for SDI applications.

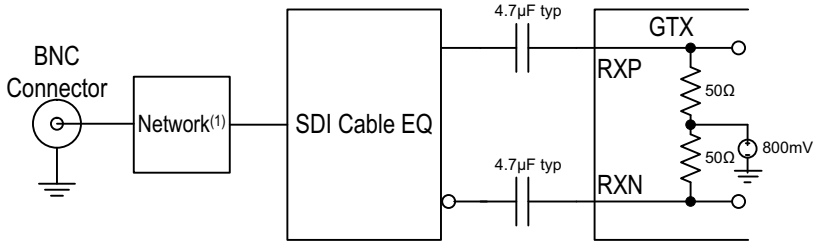

Figure 6: **Interfacing an SDI Cable Equalizer to the GTX Receiver Inputs**

Notes relevant to Figure 6:

1. Consult the SDI cable EQ manufacturer's information for the network between the SDI cable EQ and the BNC connector.

Similarly, the differential serial outputs of the GTX transmitter are connected to the inputs of an SDI cable driver, usually with AC coupling as shown in Figure 7. The cable driver converts the differential signal from the GTX transmitter into a single-ended signal with electrical characteristics meeting the SDI standards.

Important: The capacitance values of the AC coupling capacitors between the GTX TX serial outputs and the inputs of the SDI cable driver must be large enough to pass the SDI pathological signals without significant signal droop. AC coupling capacitors with values of at least 1.0 µF are required and 4.7 µF capacitors are recommended.

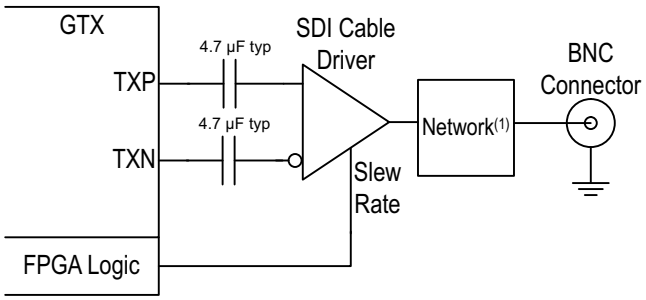

Figure 7: **Interfacing an SDI Cable Driver to the GTX Transmitter Outputs**

Notes relevant to Figure 7:

1. Consult the SDI cable driver manufacturer's information for the network between the SDI cable drive and the BNC connector.

SD-SDI Considerations

Receiving SD-SDI

The 270 Mb/s bit rate of SD-SDI is below the minimum line rate supported by the GTX RX. To receive 270 Mb/s SD-SDI, the GTX RX is used as an asynchronous oversampler to sample the SD-SDI bit stream at 11 times 270 Mb/s (2.97 gigasamples per second (GSPS)) without regard to where bit transitions occur. The CDR unit in the GTX RX is locked to the reference clock by asserting the GTX rxcdrhold input port High. This prevents the CDR from trying to lock to the slow SD-SDI signal and results in more uniform oversampling of the SD-SDI signal.

A data recovery unit (DRU), implemented in the programmable logic of the FPGA, examines the oversampled SD-SDI data from the GTX RX, determines the best sample to use for each bit, and outputs the recovered data. This DRU is not part of the SDI core, but is provided as part of this applications note's control module.

The DRU provided with this application note is described in *Clock and Data Recovery Unit based on 20-Bit-Wide Oversampled Data* [Ref 18]. That application note does describe the theory of operation of the DRU, but is not necessary for use of the DRU in the UHD-SDI reference design.

SMPTE ST 259 (the SD-SDI standard) specifies several other bit rates besides 270 Mb/s. The DRU is instantiated into the SDI control module so as to support only 11X oversampling of 270 Mb/s serial data. However, if other SD-SDI bit rates must be supported by the application, the DRU can used to receive those bit rates, as well. Because that DRU supports fractional

oversampling factors, it is possible to receive the other SD-SDI bit rates without requiring any additional RX reference clock frequencies. Note that the 540 Mb/s SD-SDI bit rate specified by SMPTE ST 344 is within the supported line rate range of the GTX transceiver and thus the GTX RX does not need to use the DRU to receive it. However, receiving the 540 Mb/s bit rate without the DRU requires a different reference clock frequency than is used for the other SDI bit rates. Thus, it is usually more convenient to use the DRU to receive the 540 Mb/s ST 344 signal using 5.5X oversampling so that the standard SDI reference clock frequency can be used. Xilinx does not have an example design supporting additional SD-SDI bit rates.

The DRU does not recover a clock and, because the CDR unit in the GTX RX is locked to its reference clock, the RXOUTCLK is not locked to the incoming bit rate in SD-SDI mode. The DRU does produce a data strobe indicating when a 10-bit data word is ready on its output. This data strobe is used by the SDI core to generate a clock enable that is asserted at a 27 MHz rate, typically with a 5/6/5/6 cadence relative to the rxoutclk clock from the GTX. The rx_ce_out output of the SDI wrapper is derived from the DRU data strobe and has the same cadence. Occasionally the cadence of the DRU data strobe and the rx_ce_out signal varies from the typical 5/6/5/6 cadence. This occurs when the DRU must make up for the slight difference between the actual SD-SDI bit rate and the frequency of the local reference clock provided to the GTX RX.

Figure 8 is a screen capture from an oscilloscope showing the 27 MHz rx_ce_out signal. The scope is triggered on the rising edge of rx_ce_out at the center of the screen. The scope is in infinite persistence mode and the waveform was allowed to accumulate for several minutes. The waveform is temperature-coded from red, indicating the most common position of the signal, to blue, indicating the least common position. The incoming SD-SDI signal that was used to create this screen capture was asynchronous to the local reference clock used by the GTX receiver. The rx_ce_out pulses on either side of the center pulse are always 5 or 6 clock cycles away from the center pulse because of the 5/6/5/6 cadence of the rx_ce_sd signal.

The two pulses at the far right and far left of the trace are nominally 11 clock cycles from the center pulse because of the 5/6/5/6 cadence. The nominal position is marked by the yellow and red pulse. And for the far right pulse, the dashed yellow vertical cursor marks the position that is 11 clock cycles from the rising edge of the center pulse. The nominal locations of the central yellow/red pulses are surrounded on either side by blue pulses indicating that, occasionally, the DRU must make the period of the rx_ce_sd cycle either 10 clock cycles or 12 clock cycles long to compensate for the frequency differences between the local reference clock and the incoming SD-SDI signal.

The SD-SDI DRU is supplied with this application note as an encrypted VHDL file. The encryption used on the DRU is compatible with most synthesis and simulation software.

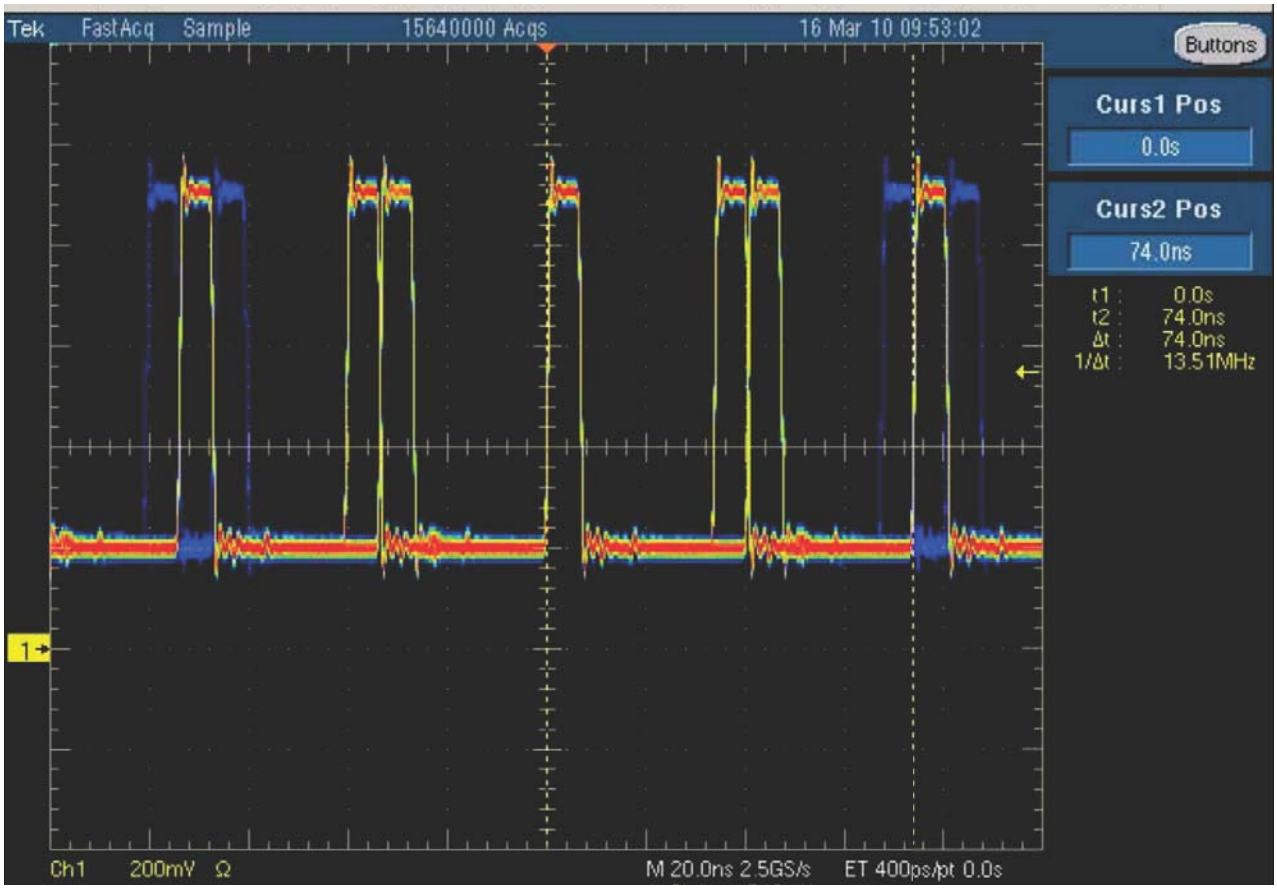

Figure 8: **Oscilloscope Capture of SD-SDI Clock Enable**

Transmitting SD-SDI

As with reception of SD-SDI, transmission of the slow 270 Mb/s SD-SDI bit rate is not directly supported by the GTX TX. To transmit the SD-SDI signal, the GTX TX is configured for a line rate of 2.97 Gb/s (one of the 3G-SDI line rates). The UHD-SDI core replicates each bit to be transmitted 11 times so that the data out of the SDI core and into the txdata port of the GTX TX contains 11 consecutive copies of each bit. The resulting signal output by the GTX TX is a valid 270 Mb/s SD-SDI signal.

Generating an SD-SDI Recovered Clock

In SD-SDI mode, the rxoutclk of the GTX RX is not really a recovered clock because the CDR unit is locked to the frequency of the reference clock, not to the SD-SDI serial stream. The only signal available that actually indicates the data rate of the incoming SD-SDI signal is the 27 MHz rx_ce_out output of the UHD-SDI wrapper.

For some video applications, particularly those that do not need to retransmit the recovered video over an SDI interface, the rx_ce_out signal might be sufficient as a recovered clock. Typically, this signal is used as a clock enable to downstream modules that are clocked with the rxoutclk from the GTX receiver. This is how the UHD-SDI datapath in the UHD-SDI core works - using the rx_ce_out signal as a clock enable.

If an actual 27 MHz recovered SD-SDI clock is required, there are several options that can be used to generate one. Refer to Application Note *Using Kintex-7 GTX Transceivers for SDI Interfaces* [Ref 17] for more details.

Automatic RX SDI Mode Detection

The UHD-SDI core can automatically determine the SDI mode (SD, HD, 3G, 6G, or 12G-SDI) of the SDI signal coming into the GTX RX. When it is not locked to the current SDI input signal, the UHD-SDI core sequences the GTX RX through the five different SDI modes until it detects recognizably good SDI data on the rxdata output port of the GTX. At that point, the UHD-SDI core indicates that the GTX CDR is locked to the SDI signal by asserting its rx_mode_locked_out port. And, it indicates which SDI mode the RX is locked to on its sdi_mode_out port.

It is important to understand that the rx_mode_locked signal is an indication of whether or not the UHD-SDI core thinks that the GTX RX is locked to the SDI signal, and nothing more. It is really just an indication of whether the UHD-SDI core's mode search state machine is still searching for the correct SDI mode or not. Because of this, rx_mode_locked should not be considered as an absolute indicator as to the locked status of the UHD-SDI RX.

When the GTX RX is not locked to the input SDI signal and the UHD-SDI core is actively controlling GTX RX in an effort to determine the correct SDI mode, the rx_mode_locked signal may briefly become asserted. This happens if the incoming data randomly appears to be a valid SAV sequence. If an SAV sequence is detected, the UHD-SDI core asserts rx_mode_locked and pause the mode search expecting more good data to be received. If, however, good data is not received within a specific timeout period, the rx_mode_locked signal is negated and the SDI mode search resumes.

The SDI mode search algorithm attempts only to lock to SDI modes that are enabled by the rx_mode_en_in port of the UHD-SDI wrapper. This 6-bit port has unary bits that enable HD-SDI (bit 0), SD-SDI (bit 1), 3G-SDI (bit 2), 6G-SDI (bit 3), 12G-SDI at 11.88 Gb/s (bit 4), and 12G-SDI at 11.88/1.001 Gb/s (bit 5). Because the GTX RX must be configured with different reference clock frequencies for the two 12G-SDI line rates, the two 12G-SDI line rates are treated as different SDI modes by the mode search algorithm. And, because there are separate enable bits on the rx_mode_en_in port, it is possible to specify that only one of the two 12G-SDI line rates should be included in the mode search. This is useful for those applications where it is undesirable to have the reference clock frequency of the QPLL frequently changed as the GTX RX scans through the two 12G-SDI line rates. For example, the use case shown in Figure 4 only supports the 11.88 Gb/s 12G-SDI line rate because only the 148.5 MHz reference clock is supplied to the QPLL. In this case, the application should set bit 5 of rx_mode_en_in Low to prevent the mode search algorithm from trying to lock in the 11.88/1.001 Gb/s 12G-SDI mode. If rx_mode_en_in is driven with a value of 6'b011111 in the use case shown in Figure 4, the SDI mode search algorithm searches for a lock in all SDI modes except the 11.88/1.001 Gb/s 12G-SDI mode.

The rx_mode_en_in port can be changed dynamically. However, if the UHD-SDI RX is already locked to a mode that becomes disabled by dynamically clearing its bit on the rx_mode_en_in port, this does not automatically kick the UHD-SDI RX out of that mode. The UHD-SDI RX remains locked in the SDI mode until the input SDI signal changes or the UHD-SDI RX is reset,

forcing the SDI mode search algorithm to try and identify the SDI mode using the new settings of the rx_mode_en_in port.

It is possible to disable the automatic SDI mode search algorithm of the UHD-SDI core. The mode search algorithm is enabled only when the rx_mode_detect_en_in port is High. If this port is Low, then the UHD-SDI RX must be told what SDI mode to operate in through the rx_forced_mode_in port. When rx_mode_detect_en_in is Low and the SDI mode search algorithm is disabled, the SDI RX is in the mode specified by the rx_forced_mode_in port and the rx_mode_locked output is always High. Thus, rx_mode_locked cannot be used as a locked indicator or a data valid indicator in this mode. When the mode search algorithm is disabled, dynamic changes on rx_forced_mode_in causes the SDI control logic to dynamically change the settings of the GTX RX as necessary for the new SDI mode.

RX Bit Rate Detection

In HD-SDI, 3G-SDI, and 6G-SDI modes, the GTX RX gives no indication of whether it is receiving an integer frame rate or a fractional frame rate SDI signal. This means that it can't tell the difference between 1.485 Gb/s and 1.485/1.001 Gb/s in HD-SDI mode, for example.

However, in 12G-SDI mode, it is possible to know whether 11.88 Gb/s or 11.88/1.001 Gb/s is being received because the reference clock to the QPLL in 12G-SDI mode must correspond to the correct line rate. If the UHD-SDI core is allowed to automatically search both 12G-SDI line rates as part of its automatic mode detection search, the core knows whether it configured the GTX RX for 11.88 Gb/s or for 11.88/1.001 Gb/s in 12G-SDI mode.

For HD-SDI, 3G-SDI, and 6G-SDI modes, the SDI control logic contains a frequency comparator that compares the frequency of the recovered clock to a reference clock of a known frequency. This logic is used to generate the rx_bit_rate_out port of the UHD-SDI wrapper to indicate whether an integer frame rate or a fractional frame rate SDI signal is being received. This frequency comparator depends on a fixed frequency reference clock on the clk_in port of the UHD-SDI wrapper. The nominal frequency of the clock driving the clk_in port must be specified by the FXDCLK_FREQ parameter. This parameter to the UHD-SDI wrapper is specified as an integer in Hertz. The frequency of the reference clock must be stable enough and must be accurately enough specified by the FXDCLK_FREQ parameter to allow the frequency comparator to distinguish between recovered clock frequencies that are just 1000ppm different. Where as in 12G mode, this "rx_bit_rate" in the example design provided, is set according to the position of the DIP switch on KCU705 board. Details of this DIP switch are provided in Example Design.

Implementing an SDI Interface in 7 Series Devices

To implement an UHD-SDI interface in a 7 series GTX FPGA, perform the following steps:

1. Generate a GTX wrapper, a GTX common wrapper and a CPLL railing module using the 7 Series FPGAs Transceivers Wizard.
2. Generate the SMPTE UHD-SDI core.
3. Instance the UHD-SDI wrapper from this application note into the application. The UHD-SDI wrapper instances and interconnects the GTX wrapper, the UHD-SDI core, and the SDI control logic. The GTX common wrapper must be instantiated separately and connected to all UHD-SDI wrappers in the same GTX Quad.
4. Apply proper timing constraints to the UHD-SDI interfaces.

Generating the GTX Wrapper

Use the 7 Series FPGAs Transceivers Wizard to generate a GTX wrappers that contains the GTXE2_COMMON and GTXE2_CHANNNEL blocks.

The GTX wrapper generated by the Wizard is actually a hierarchy of wrapper levels. The upper level wrappers contain additional reset logic that is not compatible with SDI operation. So, only the lowest level GTX wrapper file is actually useful for SDI applications. The lowest level GTX wrapper always contains a single GTXE2_CHANNNEL instance. The easiest way to generate and use the GTX wrapper is to use the Wizard to generate just a single transceiver and then instantiate the lowest level GTX wrapper multiple times in the application, once for each GTX transceiver that is used for SDI. Also, the GTX common wrapper must be instantiated as many times as necessary, once for each GTX Quad containing transceivers implementing SDI interfaces. If only the CPLL is being used for serial clocks to the GTX transceivers, then the GTX common wrapper doesn't need to be instantiated at all because it only contains the QPLL. The SDI demonstration applications supplied with this application note provide examples of how to instantiate the GTX wrapper and the GTX common wrapper.

In CPLL based 7 Series GTX designs there can be a current spike on MGTAVTT immediately after configuration. Xilinx released AR# 59294 to address this issue which instructs GTX users to hold the CPLLPD High for several reference clock pulses after configuration, only then CPLLPD can be released to proceed to normal GTX initialization. A 7 Series FPGAs Transceivers Wizard module called CPLL Railing that specifically handles the CPLLPD sequencing must be used with each active CPLL in the design.

The following information details exactly the steps required to generate the GTX wrapper using the Wizard version 3.6 from the Vivado IP catalog.

IMPORTANT: *Version 3.6 of the GTX wrapper generates GTX wrapper and GTX common wrapper files that are not completely compatible with SDI and those wrappers do require some hand editing. Instructions for editing the wrapper files are located in the Editing the GTX wrapper section of this document.*

Because the top level GTX wrapper is not used in the SDI application, it is best not to generate the GTX wrapper in the same Vivado project as the SDI application. Run Vivado and create a new project just for the purpose of generating the GTX wrapper for SDI. After the GTX wrapper is generated, only those GTX wrapper files that are needed for SDI can be added to the actual SDI Vivado project. Always specify the same 7 series FPGA device in the GTX wrapper Vivado project and in the SDI Vivado project.

After creating the GTX wrapper Vivado project, open the IP catalog. The 7 Series FPGAs Transceivers Wizard is found in the IO Interfaces folder in the top-level FPGA Features and Design folder of the Vivado IP catalog. Locate the Wizard in the IP catalog and double-click on it to launch the Wizard.

Version 3.6 of the Wizard does not contain a protocol template for 6G-SDI or 12G-SDI. However, it comes with HD-SDI and 3G-SDI presets; the 3G-SDI presets are to be used as the baseline moving forward. The instructions given in this section describe how to create a GTX wrapper with all the proper settings and ports necessary for implementing an SDI interface.

The Wizard launches with the **GT Selection** tab open as shown in Figure 9. Above the tabs is a text field called **Component Name**. The name entered here is used as the name for the GTX wrapper file and the name of the GTX component. In this example, the component name is GTXE2_CHANNNEL.

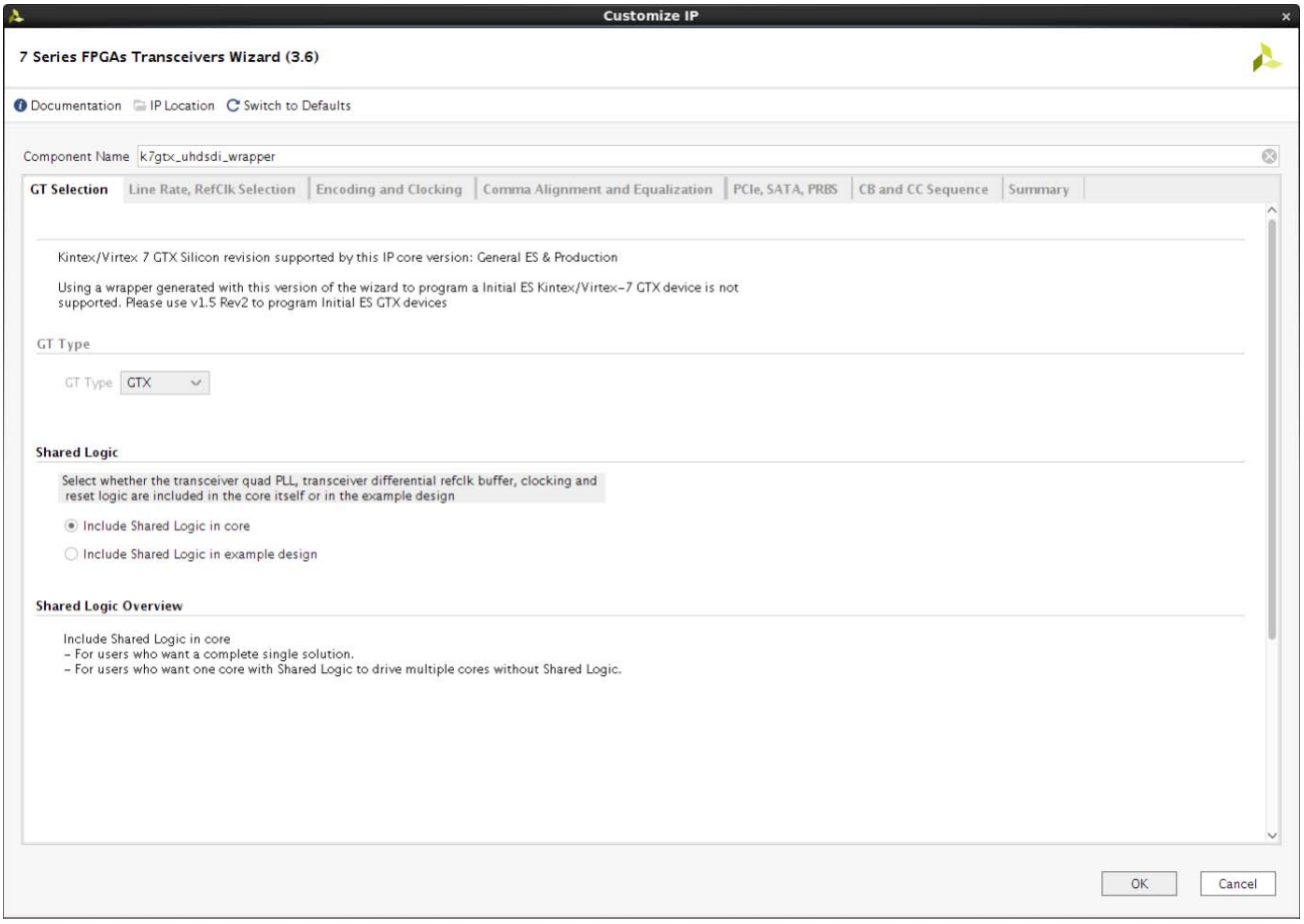

Figure 9: **7 Series Transceivers Wizard - GT Selection Tab**

Near the top of the GT selection tab type of transceiver used must be specified. Depending on the 7 series device selected for the project, GTX and/or GTH transceivers can be selected. The device selected for the Vivado project in this example only has GTX transceivers, so only GTX transceivers can be selected and the GT Type selection menu is grayed out in Figure 9.

In the Shared Logic section, select **Include Shared Logic** in example directory.

When moving from tab to tab, click on the tabs located under the **Component Name** field. Do not click the **OK** button until all tabs have been correctly setup. The **OK** button closes the Wizard.

Go to the **Line Rate, RefClk Selection** tab, shown in Figure 10. Select **3g sdi** from the **Protocol** drop down list. This sets all basic settings in the Wizard for 3G-SDI operation. This preset is to be used as baseline to change the Wizard settings for 6G-SDI and 12G-SDI applications.

The **Line Rate (Gbps)** must be set depending on the PLL used and maximum line rate to be supported. If the maximum line is 12G-SDI, then set the line rate of the RX or TX, whichever is selected to use the QPLL to 11.88 Gbps. If the maximum line rate is 6G-SDI or lower, set the line rate to 5.94 Gbps. The line rate of the other side associated to CPLL to must always be set to 5.94 Gbps. Setting the QPLL to 11.88 Gbps ensures that the Wizard will set the QPLL VCO to Upper Band which is essential for 12G-SDI operation. Set the **Reference Clock** frequency for both the TX and RX to the desired value, typically 148.5 MHz.

Do not change the line rate to 11.88/1.001 Gbps or 5.94/1.001 Gbps and the reference clock frequency to 148.5/1.001 MHz. The SDI control module takes care of switching to the 1/1.001 rates from the 1/1 rates. The control module also takes care of dynamically switching to the other line rates of 11.88 Gbps for 12G-SDI, 2.97 Gbps for 3G-SDI, 1.485 Gbps for HD-SDI and 270 Mbps for SD-SDI. The line rate specified on this tab should always be 11.88 Gbps and 5.94 Gbps. Alternative reference clock frequencies can be chosen on this tab, but only choose from those that are available in the **Reference Clock** pull down lists.

The **TX off** and **RX off** check boxes allow the creation of GTX wrappers with only transmitters (by selecting **RX off**) or only receivers (by selecting **TX off**). In this example, neither of these options is selected.

The **Quad Column** doesn't matter in this case, so just leave it to its default value.

Use Common DRP is usually not selected for SDI applications.

The bottom section of the **Line Rate, RefClk Selection** tab allows the user to choose which GTX transceivers and Quads are included in the top level GTX wrapper. It also allows the user to choose the reference clocks used by the PLLs and which PLL supplies the serial clock to each transceiver. For SDI applications always generate a GTX wrapper with a single GTX transceiver. It doesn't matter which transceiver is selected and using the single transceiver that is selected by default is the easiest.

In this example, the RX unit uses the QPLL which uses REFCLK1 Q1 as its reference clock. The TX unit uses the CPLL referenced to REFCLK0 Q1. The Wizard doesn't explicitly handle the case where TX units are dynamically switched between the QPLL and the CPLL. The SDI control module takes care of the control for this dynamic switching. But, to build a GTX wrapper with all the PLLs active and connected properly for dynamic switching of the TX between the QPLL and

the CPLL, assign the QPLL as the RX clock source and the CPLL as the TX clock source and assign different reference clocks to the QPLL and the CPLL as shown in Figure 10. In cases where the QPLL is not being used and only the CPLL is used, use the CPLL as the reference clock source to both the RX and the TX units.

Enable the **Advanced Clocking Option**.

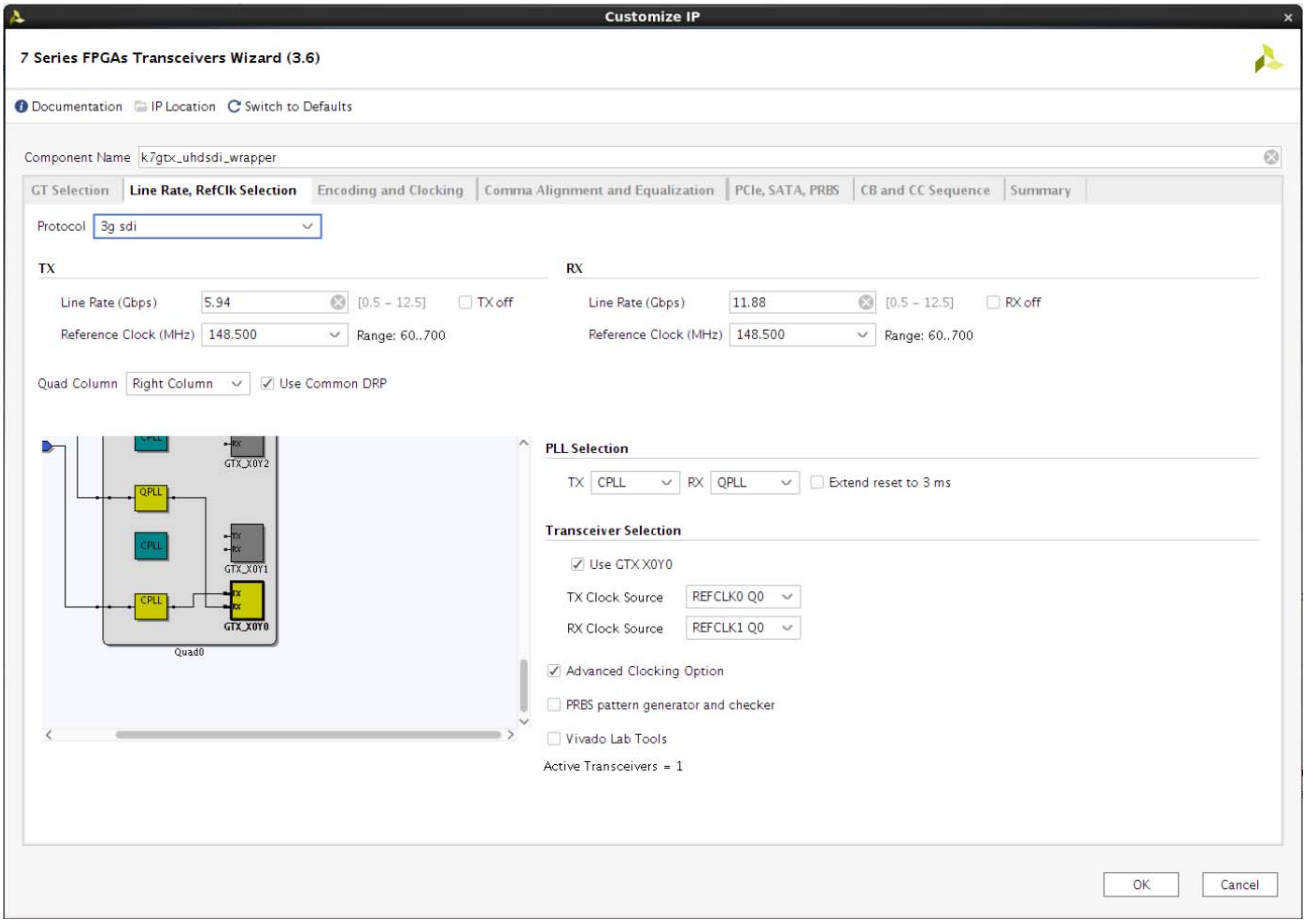

Figure 10: **7 Series Transceivers Wizard - Line Rate, RefClk Selection Tab**

Go to the **Encoding and Clocking** tab, shown in Figure 11.

For both the TX and the RX section, the **External Data Width** must be set to 40 and the **Internal Data Width** must also be set to 40. The **TX Encoding** and the **RX Decoding** must be set to **None**.

Use DRP is always selected and cannot be deselected. Set the DRP frequency to the nominal frequency of the clock connected to the GTX drpclk port.

None of the optional ports in the top section, under the DRP frequency selection, are required for SDI.

It is highly recommended that the RX and TX buffers be used for SDI applications. Thus, the **Enable TX Buffer** and **Enable RX Buffer** should be selected. The **TXUSRCLK Source** is set to TXOUTCLK and cannot be changed. However, the **RXUSRCLK Source** must be made sure to be of value RXOUTCLK, as shown in Figure 11

In the bottom Optional Ports section, the following ports are required for SDI applications: **RXSYSCLKSEL**, **RXCDRHOLD**. If the application requires that the TX units dynamically switch between the QPLL and the CPLL, then the **TXSYSCLKSEL** port is also required. It is recommended that the **TXSYSCLKSEL** port always be selected and, if dynamic switching of the TX is not required, the **TXSYSCLKSEL** port can be hard wired to select either the QPLL or the CPLL as the serial clock source.

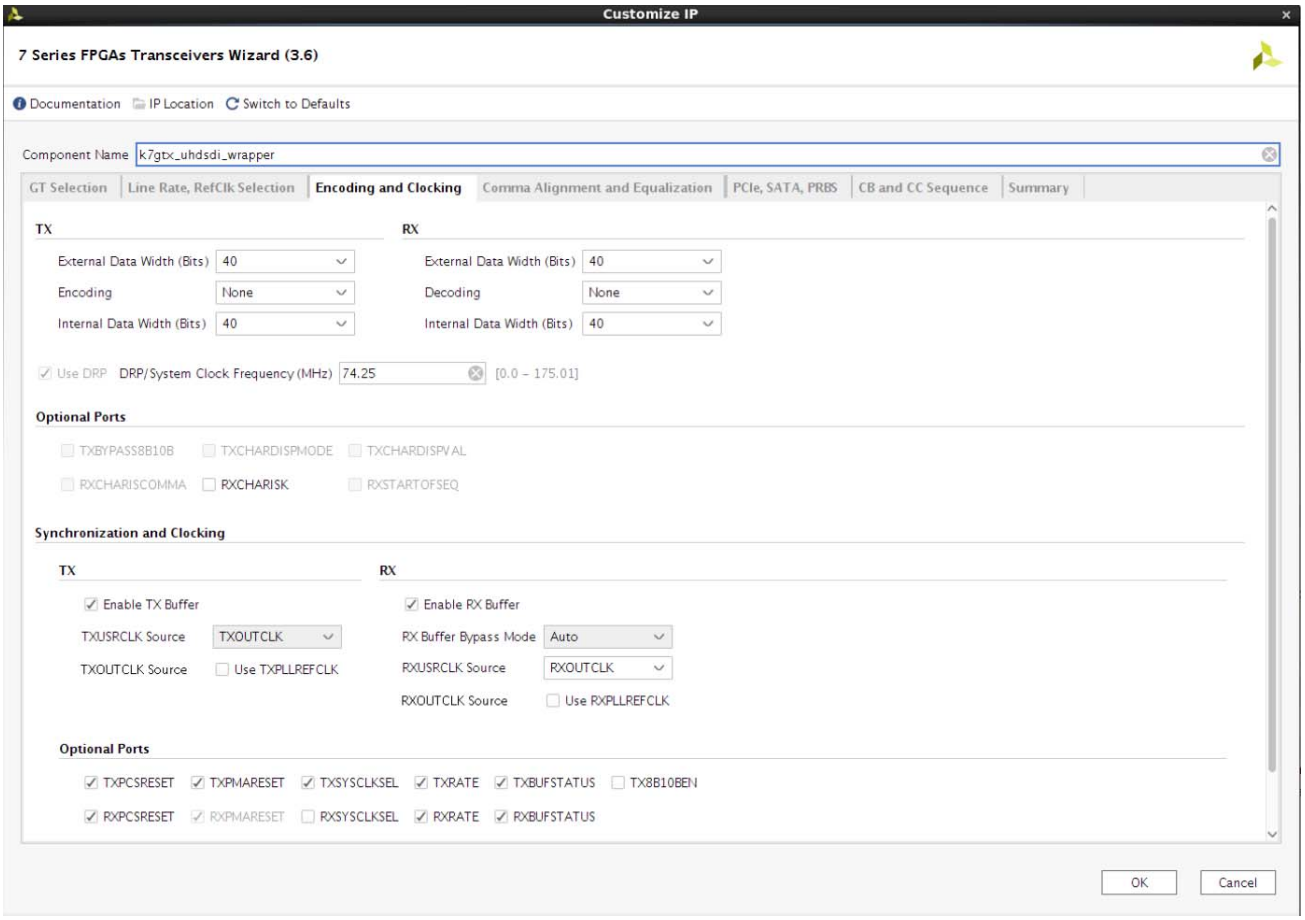

Figure 11: **7 Series Transceivers Wizard - Encoding and Clocking Tab**

Go to the **Comma Alignment and Equalization** tab, shown in Figure 12.

The settings in the **Termination and Equalization** must be changed to the values shown in Figure 12. The **Differential Swing and Emphasis Mode** must be set to **Custom, RX Equalization Mode** must be set to LPM-Auto, the **RX Termination Voltage** must be set to Programmable, and the **Trim Value** must be set to **800 mV.**

In the **Optional Ports** section, any of these ports can be enabled or disabled depending on the application requirements. The **TXPOSTCURSOR** and **TXPRECURSOR** ports can be selected if these ports are needed to improve the integrity of the signal from the TX to the external SDI cable driver.

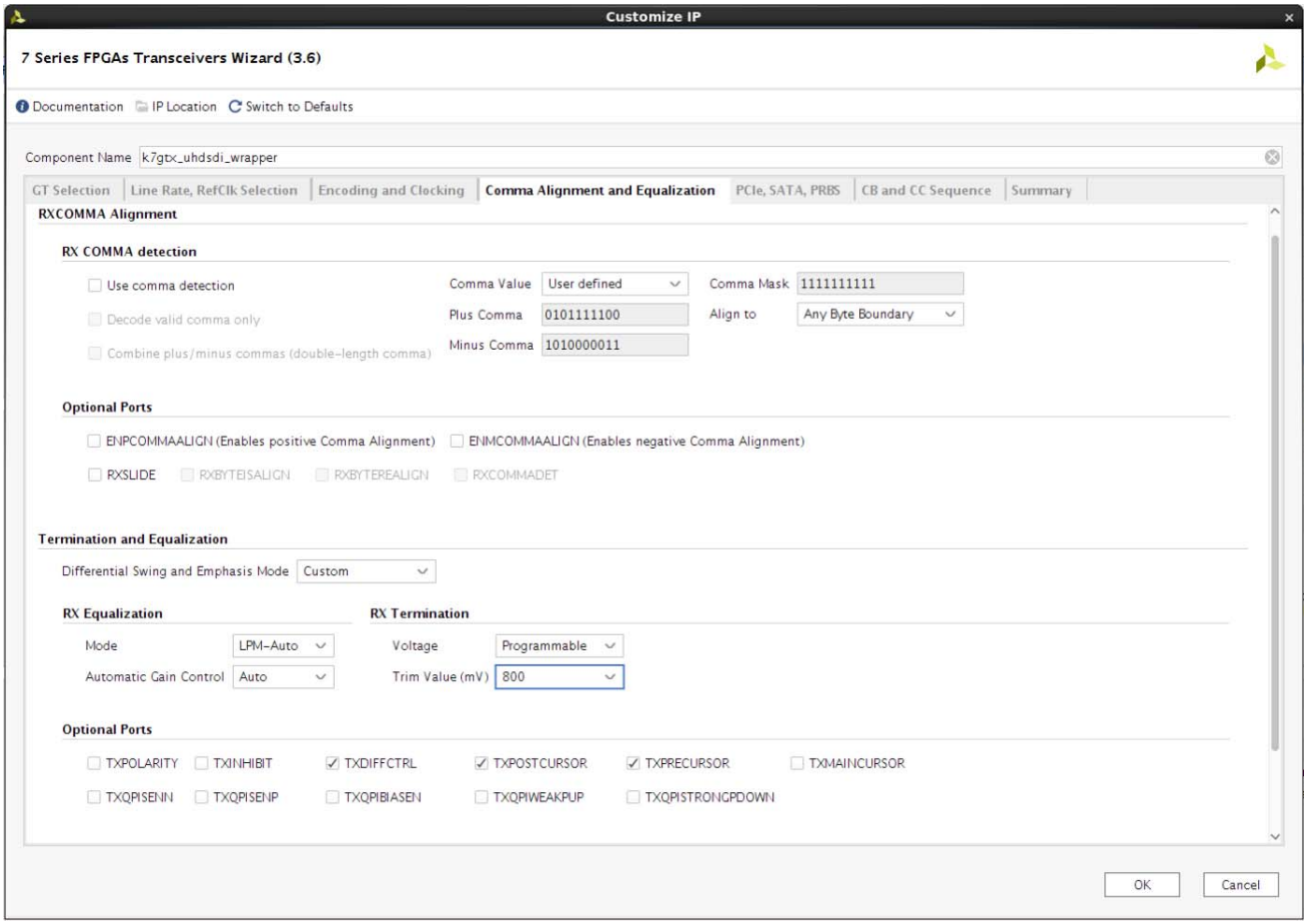

Figure 12: **7 Series Transceivers Wizard - Comma Alignment and Equalization Tab**

Move to the **PCIE, SATA, PRBS** tab, shown in Figure 13. Most of the options on this page are not relevant to SDI and should be left in their default values. There are a few ports in the Optional Ports section that can be useful for SDI applications.

The **LOOPBACK** port is selected by default. This port allows for dynamic selection of various loopback modes where the data being transmitted by the GTX TX is looped back to the GTX RX in the same transceiver. The loopback modes can be useful for debugging purposes, but generally are not used in production applications.

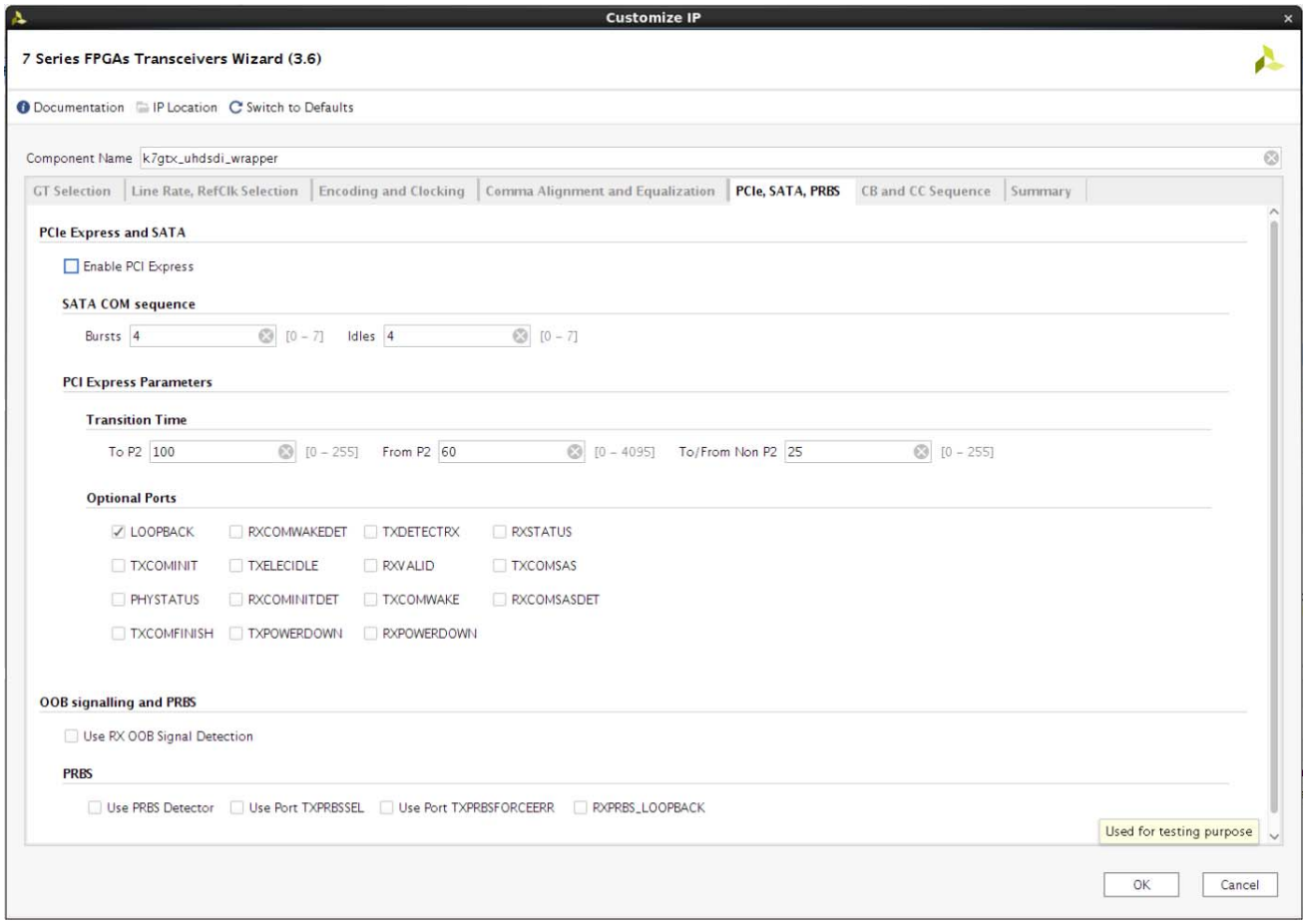

Figure 13: **7 Series Transceivers Wizard - PCIE, SATA, PRBS Tab**

At this point, all selections necessary for creating a GTX wrapper for SDI applications have been made. The **CB and CC Sequence** tab is for protocols that use channel bonding and clock correction. SDI uses neither of these. The **Summary** tab provides a summary of the selections made on the other tabs. The GTX wrapper is generated by clicking the **OK** button and then the **Generate** button when the next menu opens.

The Wizard generates various files, some that are required for an SDI application plus a lot of other example files that should not be used for SDI. The files that are used all start with the component name assigned to the GTX wrapper by the user in the Wizard. The required files are:

- <component_name>_gt.v: lowest level GTX wrapper
- <component_name>_cpll_railing: cpll railing logic
- <component_name>_common.v: common wrapper

If the Vivado project name is sdi_wrapper, Verilog is selected as the default language, and the component name given to the GTX wrapper is k7gtx_uhdsdi_wrapper, then the paths to the necessary files would be:

```
sdi_wrapper/sdi_wrapper.srcs/sources_1/ip/k7gtx_uhdsdi_wrapper/k7gtx_uhdsdi_wrapper_gt.v
sdi_wrapper/sdi_wrapper.srcs/sources_1/ip/k7gtx_uhdsdi_wrapper/k7gtx_uhdsdi_wrapper_cpll_railing
.v
sdi_wrapper/sdi_wrapper.srcs/sources_1/ip/k7gtx_uhdsdi_wrapper/k7gtx_uhdsdi_wrapper_common.v
```

The support directory and the GTX common wrapper located in the support directory aren't automatically created when you generate the GTX wrapper using the Wizard. You must right click on the SDI wrapper item in the Pop-out menu and choose the **Generate Output Products**... action as shown in Figure 14.

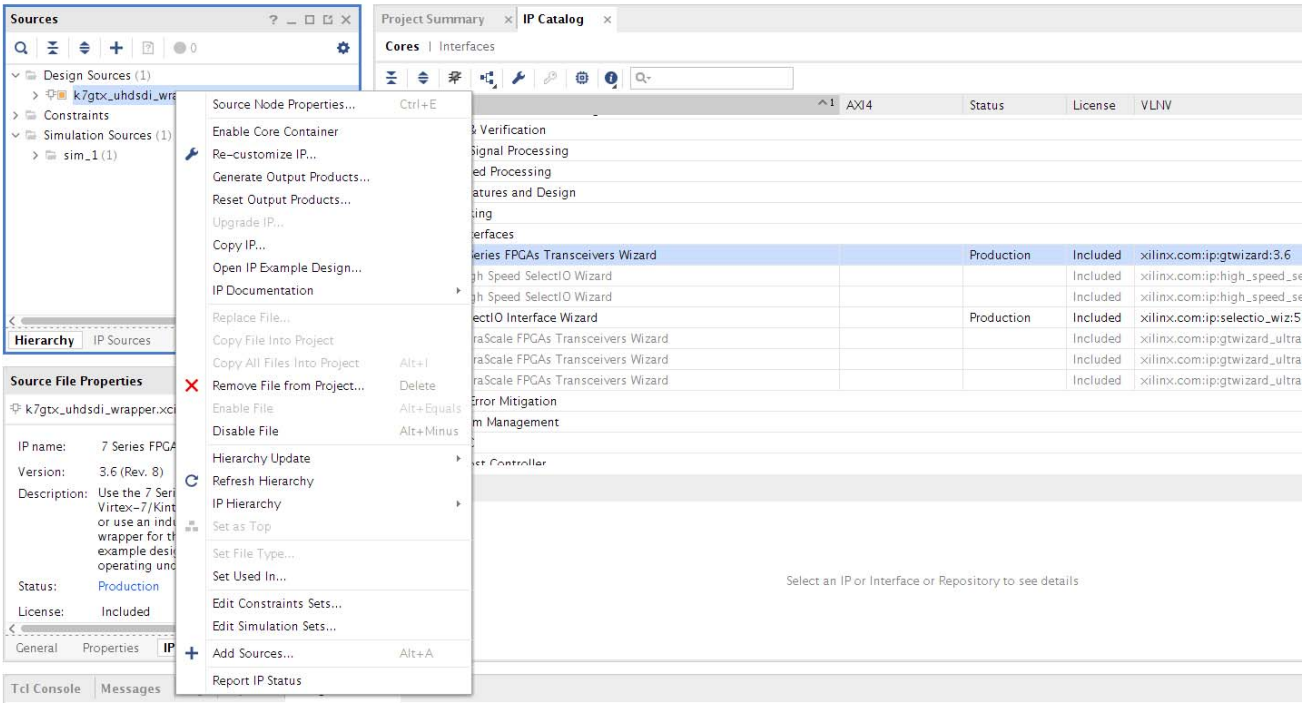

Figure 14: **Generating the Support Directory**

Editing the GTX Wrapper

Version 3.6 of the Wizard generates GTX wrapper and GTX common wrappers that require some hand editing to be compatible with SDI. This section describes the required changes to these wrapper files.

The GTX common wrapper file, named <component_name>_common.v generated by version 3.6 of the wizard has an incorrect parameter that causes the QPLL to run at the wrong frequency. The parameter is QPLL_FBDIV_TOP. The correct value of this parameter for SDI is 80 when using a reference clock of 148.5 MHz or 148.5/1.001 MHz.

The QPLL_FBDIV_TOP parameter is used to calculate the correct values of the GTXE2_COMMON primitive's QPLL_FBDIV and QPLL_FBDIV_RATIO parameters. If a reference clock frequency other than 148.5 MHz or 148.5/1.001 MHz is used, refer to the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 15] for details of how to determine the correct values for the QPLL_FBDIV and QPLL_FBDIV_RATIO parameters.

In the lowest level GTX wrapper, named <component_name>_gt.v, several changes must be made to turn off auto adaptation of the LPM equalizer in SD-SDI mode. The **rxosovrden** input port must be added in the wrapper top level ports and be connected to the RXOSOVRDEN port of the GTXE2_CHANNEL instance.

Assign the following equalizer attributes as follows:

RX_DFE_GAIN_CFG23'h0200EA

RX_DFE_H3_CFG12'b000000111110

RX_DFE_H4_CFG11'b00011011110

RX_DFE_H5_CFG11'b00011011110

RX_DFE_KL_CFG13'h00FE

RX_DFE_LPM_CFG16'h0954

RX_DFE_VP_CFG17'h03F03

By default, RXDFEAGCOVRDEN, RXDFELFOVRDEN, RXDFETAP2OVRDEN, RXDFETAP3OVRDEN, RXDFETAP4OVRDEN, RXDFETAP5OVRDEN and RXDFEUTOVRDEN ports are wired to the net named tied_to_ground_i. Change the signal wired to these ports to tied_to_vcc_i. The GTX wrapper provided with the demonstration applications supplied with this application note has these changes made so it can be used as an example.

Generate the SMPTE UHD-SDI IP

Use the Vivado IP catalog to generate the SMPTE UHD-SDI core. The SMPTE UHD-SDI core is located in the Video & Image Processing folder of the IP catalog.

The UHD-SDI core is a source code core, not a precompiled core. When the UHD-SDI core is generated, a folder is created containing the source code files for the UHD-SDI core in Verilog.

The options available when generating the UHD-SDI core is whether or not to include the error detection and handling (EDH) processor for the RX section and maximum line rate the core can support.

Choosing the Maximum Line Rate affects the maximum SDI data streams (DS) that are activated in the IP. Selecting **3G-SDI** activates 4 DS; **6G-SDI** and **12G-SDI 8DS** activates 8 DS; and **12G-SDI 16DS** activates 16 DS.

Instancing the UHD-SDI Wrapper

The UHD-SDI wrapper contains one instance of a GTX transceiver wrapper, one instance of a UHD-SDI core, and the SDI control logic. The tables and information in this section describe the UHD-SDI wrapper, its ports, and how it is used.

The UHD-SDI wrapper expects the GTX wrapper module name to be k7gtx_uhdsdi_wrapper_GT. If the GTX wrapper name is different, then the UHD-SDI wrapper must be edited appropriately. Also, if the GTX wrapper has a different set of ports than the example GTX wrapper included with the design, the instance of the GTX wrapper in the UHD-SDI wrapper must be edited.

There are 8 wrappers included in the reference design, one for each possible UHD-SDI core configuration. Instantiation and usage depend entirely upon the UHD-SDI core configuration for error free compilation. The wrappers are located in `\srcs\x7gtx_uhdsdi_wrapper` folder, this reference design uses the one in **bold**.

- x7gtx_uhdsdi_3g_wrapper
- x7gtx_uhdsdi_3g_norxedh_wrapper
- x7gtx_uhdsdi_6g_wrapper
- x7gtx_uhdsdi_6g_norxedh_wrapper
- **x7gtx_uhdsdi_12g_8s_wrapper**
- x7gtx_uhdsdi_12g_8s _norxedh_wrapper
- x7gtx_uhdsdi_12g_16s_wrapper
- x7gtx_uhdsdi_12g_16s _norxedh_wrapper

Table 2: **GTX Ports of the UHD-SDI Wrapper**

Port Name	Width	Description
clk_in	1	This is the fixed frequency clock input for various timing functions including the SDI bit rate detection frequency comparator. The minimum clock frequency here should be 10 MHz. The maximum is about 150 MHz depending on the speed grade. This clock should never stop running while the SDI interfaces are operating and should never change frequency.
drpclk_in	1	This is the clock input for the DRP port of the GTX and the SDI control logic that interfaces to the DRP port. This must be driven by a free running clock. Typically this clock input is driven by the same clock that is driving the clk_in port. The minimum clock frequency should be 10 MHz. The maximum clock frequency is the max frequency supported by the GTX DRP as described in the FPGA data sheet. This clock should never stop running while the SDI interfaces are operating and should never change frequency.
qpllclk_in	1	Connect this port to the GTX common QPLLOUTCLK_OUT port.
qpllrefclk_in	1	Connect this port to the GTX common QPLLOUTREFCLK_OUT port.
qplllock_in	1	Connect this port to the GTX common QPLLLOCK_OUT port.
cpll_refclksel_in	3	This input port selects the clock used by the CPLL. This port is connected directly to the GTX wrapper's cpllrefclksel_in port. The encoding of this port is: 000: reserved 001: cpll_gtrefclk0_in 010: cpll_gtrefclk1_in 011: cpll_northrefclk0_in 100: cpll_northrefclk1_in 101: cpll_southrefclk0_in 110: cpll_southrefclk1_in 111: not supported
cpll_northrefclk0_in	1	Northbound refclk 0 for the CPLL.
cpll_northrefclk1_in	1	Northbound refclk 1 for the CPLL.
cpll_southrefclk0_in	1	Southbound refclk 0 for the CPLL.
cpll_southrefclk1_in	1	Southbound refclk 1 for the CPLL.
cpll_gtrefclk0_in	1	This Quad's dedicated refclk 0 input.

Table 2: **GTX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
cpll_gtrefclk1_in	1	This Quad's dedicated refclk 1 input.
cplllock_out	1	This output is driven by the GTX transceiver's cplllock_out and indicates the lock status of the CPLL.
cpllreset_in	1	This input resets the GTX transceiver's CPLL. This should be driven by an appropriate combination of the rx_pll_reset_out and tx_pll_reset_out.
rxp_in	1	This is the transceiver's RXP serial input and must be connected to an input port at the top level of the design.
rxn_in	1	This is the transceiver's RXN serial input and must be connected to an input port at the top level of the design.
txp_out	1	This is the transceiver's TXP serial output and must be connected to an output port at the top level of the design.
txn_out	1	This is the transceiver's TXN serial output and must be connected to an output port at the top level of the design.

Table 3: **RX Ports of the UHD-SDI Wrapper**

Port Name	Width	Description
rx_rst_in	1	Synchronous reset input. The receiver is fully reset only if rx_rst_in is High when rx_ce_in is High on a rising edge of rx_usrclk_out.
rx_mode_detect_rst_in	1	Synchronous reset that resets only the SDI mode detect search function. The SDI mode detect search function is reset only when rx_mode_detect_rst_in is High when rx_ce_in is High on a rising edge of rx_usrclk_out.
rx_fabric_rst_out	1	This synchronous reset output is asserted High while the GTX is being reset. The user application can use this to reset any logic connected to the output of the UHD-SDI RX.
rx_usrclk_out	1	This is the main RX clock. The source of this clock is the recovered clock output (RXOUTCLK) of the GTX. The UHD-SDI wrapper has a BUFG which buffers the GTX RXOUTCLK to produce the global clock rx_usrclk_out. Unless otherwise noted, all RX ports of the UHD-SDI core are synchronous with rx_usrclk_out.
rx_gtx_full_reset_in	1	When High, this input initiates a full reset of the RX portion of the GTX. The rx_pll_reset_out signal is asserted to reset the PLL associated with the RX section and the GTX RX is reset using the gtrxreset port of the GTX. This input is synchronous with the drpclk_in.
rx_gtx_reset_in	1	When High, this input initiates a gtrxreset of the GTX. The associated PLL is not reset. This input is synchronous with the drpclk_in.
rx_refclk_stable_in	1	The user application must assert this input High when the reference clock to the PLL used by the RX is stable. Driving this input Low does not initiate a reset of the PLL. This input only keeps the reset of a PLL, initiated by the rx_gtx_full_reset_in from completing until rx_refclk_stable_in is High. This input is treated as an asynchronous input.

Table 3: **RX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
rx_pll_select_in	1	This input selects the PLL to be used as the serial clock source for the GTX RX. See the PLL Configuration and Control section for more details. This input is synchronous with the drpclk_in.
rx_pll_type_in	2	Bit 0 of this port specifies the PLL type selected when rx_pll_select_in is Low. Bit 1 of this port specifies the PLL type selected when rx_pll_select_in is High. For both bits, the CPLL is select when the bit is 0 and the QPLL is selected when the bit is 1. See the PLL Configuration and Control section for more details. This input is synchronous with the drpclk_in.
rx_pll_range_in	2	Bit 0 of this port specifies the PLL range when rx_pll_select_in is Low. Bit 1 of this port specifies the PLL range when rx_pll_select_in is High. For the CPLL or the QPLL in range 1, the bit must be Low. For the QPLL in range 2, the bit must be High. See the PLL Configuration and Control section for more details.
rx_pll_reset_out	1	This output is asserted High to reset the PLL used by the GTX RX. This output should be connected to one or both of the GTX common QPLLRESET_IN port or the UHD-SDI wrapper cpllreset_in port. This output is synchronous with the drpclk_in.
rx_mode_en_in	6	Unary enable bits for the SDI mode search where a High bit includes and a Low bit excludes a particular SDI mode from the search. Bit 0 enables HD-SDI Bit 1 enables SD-SDI Bit 2 enables 3G-SDI Bit 3 enables 6G-SDI Bit 4 enables 12G-SDI @ 11.88 Gb/s Bit 5 enables 12G-SDI @ 11.88/1.001 Gb/s
rx_mode_detect_en_in	1	This port enables the SDI mode detection feature when High. When enabled, the SDI mode detector controls the receiver to search for and lock to the incoming SDI data stream. When disabled, the user application must tell the SDI receiver what SDI mode to operate in using the rx_forced_mode_in port.
rx_forced_mode_in	3	When the rx_mode_detect_en_in input is Low, disabling the automatic SDI mode detection feature, the receiver operates in the SDI mode specified by the value on the rx_forced_mode_port_in. 000 = HD 001 = SD 010 = 3G 100 = 6G 101 = 12G 11.88 Gb/s 110 = 12G 11.88/1.001 Gb/s

Table 3: **RX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
rx_mode_out	3	This output port indicates the current SDI mode of the receiver: 000 = HD 001 = SD 010 = 3G 100 = 6G 101 = 12G 1000/1000 110 = 12G 1000/1001 When the receiver is not locked, the rx_mode_out port changes values as the receiver searches for the correct SDI mode. During this time, the rx_mode_locked_out output is Low. When the receiver detects the correct SDI mode, the rx_mode_locked_out output goes High.
rx_mode_hd_out	1	High when RX is locked in HD-SDI mode.
rx_mode_sd_out	1	High when RX is locked in SD-SDI mode.
rx_mode_3g_out	1	High when RX is locked in 3G-SDI mode.
rx_mode_6g_out	1	High when RX is locked in 6G-SDI mode.
rx_mode_12g_out	1	High when RX is locked in 12G-SDI mode (either line rate).
rx_mode_locked_out	1	High when the SDI mode search function has detected good data from the GTX and has stopped the mode search. See the PLL Configuration and Control section for more details. If the rx_mode_detect_en_in port is Low, this output is always High.
rx_bit_rate_out	1	This output indicates which bit rate is being received in HD-SDI, 3G-SDI, 6G-SDI, and 12G-SDI modes as shown below: HD-SDI mode: rx_bit_rate = 0: Bit rate = 1.485 Gb/s rx_bit_rate = 1: Bit rate = 1.485/1.001 Gb/s 3G-SDI mode: rx_bit_rate = 0: Bit rate = 2.97 Gb/s rx_bit_rate = 1: Bit rate = 2.97/1.001 Gb/s 6G-SDI mode: rx_bit_rate = 0: Bit rate = 5.94 Gb/s rx_bit_rate = 1: Bit rate = 5.94/1.001 Gb/s 12G-SDI mode: rx_bit_rate = 0: Bit rate = 11.88 Gb/s rx_bit_rate = 1: Bit rate = 11.88/1.001 Gb/s
rx_t_locked_out	1	A High on this output indicates that the transport format detection logic has identified the SDI transport format.
rx_t_family_out	4	This output indicates which family of video signals is being used as the transport of the SDI interface. This output is only valid when rx_t_locked_out is High. This port does not necessarily identify the video format of the picture being transported. It only identifies the transport characteristics. See Table 8 for encoding of this port.

Table 3: **RX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
rx_t_rate_out	4	This output indicates the frame rate of the transport. This is not necessarily the same as the frame rate of the actual picture. This output is only valid when rx_t_locked_out is High. See Table 9 for details on the encoding of this port.
rx_t_scan_out	1	This output indicates whether the transport is interlaced (Low) or progressive (High). This is not necessarily the same as the scan mode of the actual picture. This output is only valid when rx_t_locked_out is High.
rx_level_b_3g_out	1	In 3G-SDI mode, this output is asserted High when the input signal is level B and Low when it is level A. This output is only valid when rx_mode_3g_out is High.
rx_active_streams_out	3	This port indicates the number of data streams that are active for the current video format being received. The number of active data streams is $2^{rx\_active\_streams\_out}$. 000: 1 active stream 001: 2 active streams 010: 4 active streams 011: 8 active streams 100: 16 active streams Other values are reserved.
rx_ce_out	1	This is the RX clock enable output. This clock enable is valid in all SDI modes. In SD mode, rx_ce_out has a nominal 5/6/5/6 cadence. In HD and 3GA modes, rx_ce_out is always High. In 3GB mode, rx_ce_out has a 50% duty cycle. In 6G, the duty cycle can by 100% or 50% depending on how may data streams are interleaved onto the signal. In 12G, the duty cycle can be 50% or 25% depending on how many data streams are interleaved onto the signal.
rx_line_0_out	11	Captured line number from data stream 1 is output here. Not valid in SD-SDI mode.
rx_line_1_out	11	Captured line number from data stream 3 is output here. Only valid if 4 or more data streams are active.
rx_line_2_out	11	Captured line number from data stream 5 is output here. Only valid if 8 or more data streams are active.
rx_line_3_out	11	Captured line number from data stream 7 is output here. Only valid if 8 or more data streams are active.
rx_line_4_out	11	Captured line number from data stream 9 is output here. Only valid if 16 data streams are active.
rx_line_5_out	11	Captured line number from data stream 11 is output here. Only valid if 16 data streams are active.
rx_line_6_out	11	Captured line number from data stream 13 is output here. Only valid if 16 data streams are active.
rx_line_7_out	11	Captured line number from data stream 15 is output here. Only valid if 16 data streams are active.
rx_st352_0_out	32	The ST 352 payload ID packet data bytes captured from data stream 1 are output here.
rx_st352_0_valid_out	1	High when rx_st352_0 is valid.

Table 3: **RX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
rx_st352_1_out	32	The ST 352 payload ID packet data bytes captured from data stream 3 are output here. In 3G-SDI level A mode, the ST 352 payload ID packet data bytes from data stream 2 are output here.
rx_st352_1_valid_out	1	High when rx_st352_1 is valid.
rx_st352_2_out	32	The ST 352 payload ID packet data bytes captured from data stream 5 are output here.
rx_st352_2_valid_out	1	High when rx_st352_2 is valid.
rx_st352_3_out	32	The ST 352 payload ID packet data bytes captured from data stream 7 are output here.
rx_st352_3_valid_out	1	High when rx_st352_3 is valid.
rx_st352_4_out	32	The ST 352 payload ID packet data bytes captured from data stream 9 are output here.
rx_st352_4_valid_out	1	High when rx_st352_4 is valid.
rx_st352_5_out	32	The ST 352 payload ID packet data bytes captured from data stream 11 are output here.
rx_st352_5_valid_out	1	High when rx_st352_5 is valid.
rx_st352_6_out	32	The ST 352 payload ID packet data bytes captured from data stream 13 are output here.
rx_st352_6_valid_out	1	High when rx_st352_6 is valid.
rx_st352_7_out	32	The ST 352 payload ID packet data bytes captured from data stream 15 are output here.
rx_st352_7_valid_out	1	High when rx_st352_7 is valid.
rx_crc_err_out	16	These 16 bits are the CRC error indicator for each data stream output. Bit 0 is the CRC error indicator for data stream 1, bit 1 for data stream 2, etc. When a CRC is detected on a line, the CRC error bit corresponding to that data stream becomes asserted starting a few clock cycles after the last CRC word is output on the data stream ports following the EAV that ends the line containing the error. The CRC error bit remains asserted for one line time. These bits are not valid in SD-SDI mode.
rx_ds1_out	10	Data stream 1 output. In SD mode this is interleaved Y/C. In HD and 3G level A modes, this is the Y channel. In 3G level B mode, this is the link A Y channel. In 6G and 12G modes, this is ds1.
rx_ds2_out	10	Data stream 2 output. Not used in SD mode. In HD and 3G level A modes, this is the C channel. In 3G level B mode, this is the link A C channel. In 6G and 12G modes, this is ds2.
rx_ds3_out	10	Data stream 3 output. Not used in SD, HD, and 3G level A modes. In 3G level B mode, this is the link B Y channel. In 6G and 12G modes this is ds3.
rx_ds4_out	10	Data stream 4 output. Not used in SD, HD, and 3G level A modes. In 3G level B mode, this is the link B C channel. In 6G and 12G modes this is ds4.
rx_ds5_out	10	Data stream 5 output. Only used in 6G and 12G modes.
rx_ds6_out	10	Data stream 6 output. Only used in 6G and 12G modes.

Table 3: **RX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
rx_ds7_out	10	Data stream 7 output. Only used in 6G and 12G modes.
rx_ds8_out	10	Data stream 8 output. Only used in 6G and 12G modes.
rx_ds9_out	10	Data stream 9 output. Only used in 12G mode when 16 data streams are active.
rx_ds10_out	10	Data stream 10 output. Only used in 12G mode when 16 data streams are active.
rx_ds11_out	10	Data stream 11 output. Only used in 12G mode when 16 data streams are active.
rx_ds12_out	10	Data stream 12 output. Only used in 12G mode when 16 data streams are active.
rx_ds13_out	10	Data stream 13 output. Only used in 12G mode when 16 data streams are active.
rx_ds14_out	10	Data stream 14 output. Only used in 12G mode when 16 data streams are active.
rx_ds15_out	10	Data stream 15 output. Only used in 12G mode when 16 data streams are active.
rx_ds16_out	10	Data stream 16 output. Only used in 12G mode when 16 data streams are active.
rx_eav_out	1	This output is asserted High when the XYZ word of an EAV is present on the data stream output ports.
rx_sav_out	1	This output is asserted High when the XYZ word of an SAV is present on the data stream output ports.
rx_trs_out	1	This output is asserted High while the four consecutive words of any EAV or SAV are present on the data stream output ports, from the 3FF word through the XYZ word.
rx_edh_errcnt_en_in	16	This input controls which EDH error conditions increments the rx_edh_errcnt_out counter. See Table 5 for encoding of this port.
rx_edh_clr_errcnt_in	1	When High, this input clears the rx_edh_errcnt_out counter. This input port must be High during the same clock cycle when rx_ce _out is also High to clear the error counter.
rx_edh_ap_out	1	This output is asserted High when the active picture CRC calculated for the previous field does not match the AP CRC value in the EDH packet.
rx_edh_ff_out	1	This output is asserted High when the full field CRC calculated for the previous field does not match the full field CRC value in the EDH packet.
rx_edh_anc_out	1	This output is asserted High when an ancillary data packet checksum error is detected.
rx_edh_ap_flags_out	5	The active picture error flag bits from the most recently received EDH packet are output on this port. See Table 6 for encoding of this port.
rx_edh_ff_flags_out	5	The full frame error flag bits from the most recently received EDH packet are output on this port. See Table 6 for encoding of this port.
rx_edh_anc_flags_out	5	The ancillary error flag bits from the most recently received EDH packet are output on this port. See Table 6 for encoding of this port.
rx_edh_packet_flags_out	4	This port outputs four error flags related to the most recently received EDH packet. See Table 7 for encoding of this port.

Table 3: **RX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
rx_edh_errcnt_out	16	This is the SD-SDI EDH error counter. It increments once per field when any of the error conditions enabled by the rx_edh_err_en_in port occur during that field.
rx_change_done_out	1	A High on this output indicates that the most recent RX reset or dynamic SDI mode change sequence completed successfully. This output is synchronous with drpclk_in.
rx_change_fail_out	1	A High on this output indicates that the most recent RX reset or dynamic SDI mode change sequence failed. This output is synchronous with drpclk_in.
rx_change_fail_code_out	3	If rx_change_fail_out is High, this port indicates the reason for the failure. See Table 11 for encoding of this port. This output is synchronous with drpclk_in.

Table 4: **TX Ports of the UHD-SDI Wrapper**

Port Name	Width	Description
tx_rst_in	1	This is a synchronous reset input. It resets the transmitter when High. To fully reset the transmitter, the tx_ce_in, tx_sd_ce_in, and tx_edh_ce_in inputs must be High when tx_rst_in is asserted.
tx_fabric_rst_out	1	This is a synchronous reset output. It is asserted whenever the GTX TX is being reset. This can be used to reset any logic in the user application that drives the UHD-SDI TX.
tx_usrclk_out	1	This is the main TX clock. The source of this clock is the TXOUTCLK output of the GTX. The UHD-SDI wrapper has a BUFG which buffers the GTX TXOUTCLK to produce the global clock tx_usrclk_out. Unless otherwise noted, all TX ports of the UHD-SDI core are synchronous with tx_usrclk_out.
tx_gtx_full_reset_in	1	When High, this input initiates a full reset of the TX portion of the GTX. The tx_pll_reset_out signal is asserted to reset the PLL associated with the TX section and the GTX TX is reset using the gttxreset port of the GTX. This input is synchronous with the drpclk_in.
tx_gtx_reset_in	1	When High, this input initiates a gttxreset of the GTX. The associated PLL is not reset. This input is synchronous with the drpclk_in.
tx_refclk_stable_in	1	The user application must assert this input High when the reference clock to the PLL used by the TX is stable. Driving this input Low does not initiate a reset of the PLL. This input only keeps the reset of a PLL, initiated by the tx_gtx_full_reset_in from completing until tx_refclk_stable_in is High. This input is treated as an asynchronous input.
tx_pll_select_in	1	This input selects the PLL to be used as the serial clock source for the GTX TX. See the PLL Configuration and Control section for more details. This input is synchronous with the drpclk_in.
tx_pll_type_in	2	Bit 0 of this port specifies the PLL type selected when tx_pll_select_in is Low. Bit 1 of this port specifies the PLL type selected when tx_pll_select_in is High. For both bits, the CPLL is select when the bit is 0 and the QPLL is selected when the bit is 1. See the PLL Configuration and Control section for more details. This input is synchronous with the drpclk_in.

Table 4: **TX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
tx_pll_range_in	2	Bit 0 of this port specifies the PLL range when tx_pll_select_in is Low. Bit 1 of this port specifies the PLL range when tx_pll_select_in is High. For the CPLL or the QPLL in range 1, the bit must be Low. For the QPLL in range 2, the bit must be High. See the PLL Configuration and Control section for more details. This input is synchronous with the drpclk_in.
tx_pll_reset_out	1	This output is asserted High to reset the PLL used by the GTX TX. This output should be connected to one or both of the GTX common QPLLRESET_IN port or the UHD-SDI wrapper cpllreset_in port. This output is synchronous with the drpclk_in.
tx_ce_in	1	This is the clock enable input for the main portion of the transmitter data path. It must always be High in SD, HD, and 3G level A modes. In 3G level B mode, it must have a 50% duty cycle. In 6G and 12G modes, it must have a 100% duty cycle when 4 streams are interleaved, 50% duty cycle when 8 streams are interleaved, and 25% duty cycle when all 16 data streams are interleaved.
tx_sd_ce_in	1	This is the clock enable for SD-SDI mode. It must have exactly a 5/6/5/6 cadence in SD-SDI mode and must be High in all other modes.
tx_edh_ce_in	1	This is the clock enable for the TX EDH processor. In SD-SDI mode, it must be exactly equal to the tx_sd_ce_in port with its 5/6/5/6 cadence. It must be phase aligned with tx_sd_ce_in. In all other modes, this input can be driven Low to reduce the power that consumed by the EDH processor.
tx_mode_in	3	This input port is used to select the transmitter SDI mode: 000 = HD 001 = SD 010 = 3G 100 = 6G 101 = 12G All other values are reserved.
tx_insert_crc	1	When this input is High, the transmitter generates and insert CRC values into the data streams for each video line in all modes except SD-SDI. When this input is Low, CRC values are not inserted into the data streams. This input is ignored in SD-SDI mode.
tx_insert_ln	1	When this input is High, the transmitter inserts line numbers into all active data streams after the EAV of each video line. The line numbers must be supplied on the tx_line_ch_*n*_in input ports of all active data stream pairs. When this input is Low, line numbers are not inserted. This input is ignored in SD-SDI mode.
tx_insert_st352	1	When this input is High, ST 352 packets are inserted into the data streams, otherwise the packets are not inserted. ST 352 packets are mandatory in 3G, 6G, and 12G modes and optional in HD and SD modes.
tx_overwrite_st352	1	If this input is High, ST 352 packets already present in the data streams are overwritten. If this input is Low, existing ST 352 packets are not overwritten.

Table 4: **TX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
tx_insert_edh	1	When this input is High, the transmitter generates and inserts EDH packets into every field in SD-SDI mode. When this input is Low, EDH packets are not inserted. This input is ignored in all modes except SD-SDI mode.
tx_mux_pattern_in	3	This specifies the data stream interleaving pattern to be used. 000 = SD, HD, and 3G level A 001 = 3G level B 010 = 8 stream interleave in 6G and 12G modes 011 = 4 stream interleave in 6G mode 100 = 16 stream interleave in 12G mode
tx_insert_sync_bit_in	1	In 6G and 12G modes, when this port is High, the sync bit insertion function is enabled for run length mitigation. For compliance with the ST 2081-1 and ST 2082-1 standards, sync bit insertion must be enabled. However, some early implementations of 6G-SDI and 12G-SDI receivers do not support sync bit insertion and when transmitting signals to those devices, sync bit insertion can be disabled by setting this port Low.
tx_line_0_in	11	Current line number for data streams 1 & 2.
tx_line_1_in	11	Current line number for data streams 3 & 4.
tx_line_2_in	11	Current line number for data streams 5 & 6.
tx_line_3_in	11	Current line number for data streams 7 & 8.
tx_line_4_in	11	Current line number for data streams 9 & 10.
tx_line_5_in	11	Current line number for data streams 11 & 12.
tx_line_6_in	11	Current line number for data streams 13 & 14.
tx_line_7_in	11	Current line number for data streams 15 & 16.
tx_st352_line_f1_in	11	The ST 352 packets are inserted into the HANC space of the line number specified by this input port. For interlaced video, this input port specifies a line number in field 1. For progressive video, this specifies the only line in the frame where the packets are inserted. The input value must be valid during the entire HANC interval. If tx_insert_st352_in is Low, this input is ignored.
tx_st352_line_f2_in	11	For interlace video, ST 352 packets are inserted on the line number in field 2 indicated by this value. For progressive video, this input port must be disabled by driving the tx_st352_f2_en_in port Low. The input value on this port must be valid during the entire HANC interval. This port is ignored if either tx_insert_st352_in or tx_st352_f2_en_in are Low.
tx_st352_f2_en_in	1	This input controls whether or not ST 352 packets are inserted on the line indicated by tx_vpid_line_f2_in. For interlaced video, this input must be High if ST 352 packet insertion is enabled. For progressive video, this input must be Low if ST 352 packet insertion is enabled. If ST 352 packet insertion is disabled (tx_insert_st352_in = Low), this port is ignored.
tx_st352_data_0_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds1 when tx_insert_st352_in is High. The data bytes are ordered like this: {byte4, byte3, byte2, byte1}.

Table 4: **TX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
tx_st352_data_1_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds3 when tx_insert_st352_in is High. In 3G-SDI level A mode, this port supplies the data bytes inserted into the ST 352 packet of ds2.
tx_st352_data_2_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds5 when tx_insert_st352_in is High.
tx_st352_data_3_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds7 when tx_insert_st352_in is High.
tx_st352_data_4_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds9 when tx_insert_st352_in is High.
tx_st352_data_5_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds11 when tx_insert_st352_in is High.
tx_st352_data_6_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds13 when tx_insert_st352_in is High.
tx_st352_data_7_in	32	This port supplies the four data bytes of the ST 352 packet inserted into ds15 when tx_insert_st352_in is High.
tx_ds1_in	10	Data stream 1 input: SD=Y/C, HD=Y, 3GA=DS1(Y), 3GB=AY, 6G/12G=DS1.
tx_ds2_in	10	Data stream 2 input: HD=C, 3GA=DS2(C), 3GB=AC, 6G/12G=DS2.
tx_ds3_in	10	Data stream 3 input: 3GB=BY, 6G/12G=DS3.
tx_ds4_in	10	Data stream 4 input: 3GB=BC, 6G/12G=DS4.
tx_ds5_in	10	Data stream 5 input: 6G/12G=DS5.
tx_ds6_in	10	Data stream 6 input: 6G/12G=DS6.
tx_ds7_in	10	Data stream 7 input: 6G/12G=DS7.
tx_ds8_in	10	Data stream 8 input: 6G/12G=DS8.
tx_ds9_in	10	Data stream 9 input: 12G=DS9.
tx_ds10_in	10	Data stream 10 input: 12G=DS10.
tx_ds11_in	10	Data stream 11 input: 12G=DS11.
tx_ds12_in	10	Data stream 12 input: 12G=DS12.
tx_ds13_in	10	Data stream 13 input: 12G=DS13.
tx_ds14_in	10	Data stream 14 input: 12G=DS14.
tx_ds15_in	10	Data stream 15 input: 12G=DS15.
tx_ds16_in	10	Data stream 16 input: 12G=DS16.
tx_ds1_st352_out	10	This is the data stream 1 (DS1) output data stream after the ST 352 packet insertion module. The data stream is output at this point for the application to embed other ANC data.
tx_ds2_st352_out	10	This is the DS2 output data stream for ANC insertion.
tx_ds3_st352_out	10	This is the DS3 output data stream for ANC insertion.
tx_ds4_st352_out	10	This is the DS4 output data stream for ANC insertion.
tx_ds5_st352_out	10	This is the DS5 output data stream for ANC insertion.
tx_ds6_st352_out	10	This is the DS6 output data stream for ANC insertion.

Table 4: **TX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
tx_ds7_st352_out	10	This is the DS7 output data stream for ANC insertion.
tx_ds8_st352_out	10	This is the DS8 output data stream for ANC insertion.
tx_ds9_st352_out	10	This is the DS9 output data stream for ANC insertion.
tx_ds10_st352_out	10	This is the DS10 output data stream for ANC insertion.
tx_ds11_st352_out	10	This is the DS11 output data stream for ANC insertion.
tx_ds12_st352_out	10	This is the DS12 output data stream for ANC insertion.
tx_ds13_st352_out	10	This is the DS13 output data stream for ANC insertion.
tx_ds14_st352_out	10	This is the DS14 output data stream for ANC insertion.
tx_ds15_st352_out	10	This is the DS15 output data stream for ANC insertion.
tx_ds16_st352_out	10	This is the DS16 output data stream for ANC insertion.
tx_ds1_anc_in	10	Data stream 1 (DS1) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds2_anc_in	10	Data stream 2 (DS2) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds3_anc_in	10	Data stream 3 (DS3) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds4_anc_in	10	Data stream 4 (DS4) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds5_anc_in	10	Data stream 5 (DS5) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds6_anc_in	10	Data stream 6 (DS6) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds7_anc_in	10	Data stream 7 (DS7) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds8_anc_in	10	Data stream 8 (DS8) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds9_anc_in	10	Data stream 9 (DS9) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds10_anc_in	10	Data stream 10 (DS10) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds11_anc_in	10	Data stream 11 (DS11) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds12_anc_in	10	Data stream 12 (DS12) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds13_anc_in	10	Data stream 13 (DS13) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds14_anc_in	10	Data stream 14 (DS14) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds15_anc_in	10	Data stream 15 (DS15) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.
tx_ds16_anc_in	10	Data stream 16 (DS16) input from the application's ANC inserter. This port is only used if the tx_use_anc_in port is High.

Table 4: **TX Ports of the UHD-SDI Wrapper *(Cont'd)***

Port Name	Width	Description
tx_user_anc_in	1	When Low, the data streams out of the ST352 packet insertion function are routed internally to the TX output channels. When High, the TX output channels accept data streams from the tx_ds[16:1]_anc_in ports.
tx_ce_align_err_out	1	This output indicates problems with the 5/6/5/6 clock cycle cadence of the tx_sd_ce_in input in SD-SDI mode. In SD-SDI mode, the tx_sd_ce_in signal must follow a regular 5/6/5/6 clock cycle cadence. If it does not, the SD-SDI serial stream is formed incorrectly. The tx_ce_align_err_out output goes High if the cadence is incorrect. This port is only valid in SD-SDI mode.
tx_change_done_out	1	A High on this output indicates that the most recent TX reset or dynamic SDI mode change sequence completed successfully. This output is synchronous with drpclk_in.
tx_change_fail_out	1	A High on this output indicates that the most recent TX reset or dynamic SDI mode change sequence failed. This output is synchronous with drpclk_in.
tx_change_fail_code_out	3	If tx_change_fail_out is High, this port indicates the reason for the failure. See Table 11 for encoding of this port. This output is synchronous with drpclk_in.

SD-SDI EDH Error Detection

The UHD-SDI receiver optionally contains an EDH processor that checks the SD-SDI signal for errors. This EDH processor does not update EDH packets in the SD-SDI stream. It simply reports any errors found and also captures the error flags from each EDH packet. The receiver EDH processor can be set to either include in the core or not or by using the INCLUDE_RX_EDH_PROCESSOR parameter to the UHD-SDI wrapper. The EDH processor has a 16-bit counter which counts the number of fields that have errors. The current error count is output on the rx_edh_errcnt_in port. The counter can be cleared by asserting rx_edh_clr_errcnt_in High. You can specify which types of errors are counted using the rx_edh_errcnt_en_in port. This port has 16 unary bits which enable and disable 16 different error types. Any bit that is High enables the corresponding error. When this type of error is detected, the error counter increments. Any bit that is Low disables the corresponding error. Table 5 shows the encoding of the bits on the rx_edh_errcnt_en_in port.

Table 5: **Encoding of rx_edh_errcnt_en_in Port**

Bit #	Error
0	ANC EDH error
1	ANC EDA error
2	ANC IDH error
3	ANC IDA error
4	ANC UES error
5	FF EDH error
6	FF EDA error
7	FF IDH error

Table 5: **Encoding of rx_edh_errcnt_en_in Port *(Cont'd)***

Bit #	Error
8	FF IDA error
9	FF UES error
10	AP EDH error
11	AP EDA error
12	AP IDH error
13	AP IDA error
14	AP UES error
15	EDH packet checksum-error

The ANC error conditions occur when there are errors in the ancillary data packets. The FF error conditions occur when there are errors in the full field. And, the AP error conditions occur when there are errors in the active portion of the picture. The EDH packet checksum error indicates a checksum error found within the EDH packet itself.

The ANC, FF, and AP error condition sets each have five individual error flags, described below. All flags are asserted High to indicate an error condition. For a complete description of the EDH, EDA, IDH, IDA, and UES error flags in the EDH packet, refer to the SMPTE RP 165 document available from SMPTE.

- EDH error: This error condition occurs when the EDH processor detects a CRC error (checksum error for ANC packets) in a field.
- EDA error: This error condition occurs when the EDA or EDH flags of the received EDH packet are asserted.
- IDH error: This error condition is not supported by the EDH processor.
- IDA error: This error condition occurs when the IDA or IDH flags of the received EDH packet are asserted.
- UES error: This error condition occurs when the UES flag in the received EDH packet is asserted.

The actively computed EDH errors for the ANC, AP, and FF are also output on the rx_edh_anc_out, rx_edh_ap_out, and rx_edh_ff_out ports, respectively. Thus, the rx_edh_anc_out port is asserted whenever a checksum error is detected in an ancillary data packet. The rx_edh_ap_out port is asserted when the calculated active picture CRC does not match the AP CRC in the EDH packet. And, the rx_edh_ff_out port is asserted when the calculated full field CRC does not match the FF CRC in the EDH packet.

The EDH processor also outputs the ANC, AP, and FF flags from the EDH packet on the rx_edh_anc_flags_out, rx_edh_ap_flags_out, and rx_edh_ff_flags_out ports, respectively. These output ports are exact copies of the flags found in the last received EDH packet. This means they differ from the actively computed error conditions shown above. For example, the EDH flag (bit 0) of the rx_edh_ap_flags_out port indicates that the AP EDH flag is set in the last received EDH packet. But, the rx_edh_ap_out port indicates that the active picture CRC calculated locally by the EDH processor does not match the AP CRC value in the EDH packet. The

rx_edh_anc_flags_out, rx_edh_ap_flags_out, and rx_edh_ff_flags_out ports are each 5 bits wide and are encoded as shown in Table 6.

Table 6: **Encoding of rx_edh_anc_flags_out, rx_edh_ap_flags_out, and rx_edh_ff_flags_out Ports**

Bit #	Error
0	EDH
1	EDA
2	IDH
3	IDA
4	UES

The EDH processor also produces four error flags related to the format and contents of the EDH packet. These error flags are output on the rx_edh_packet_flags_out port. The encoding of this port is shown in Table 7.

Table 7: **Encoding of rx_edh_packet_flags_out Port**

Bit #	Error
0	EDH packet is missing
1	Parity error in user data words of EDH packet
2	Checksum error in EDH packet
3	Format error in EDH packet – such as invalid data count

Transport Format Detection

The UHD-SDI receiver has transport format detector function. This function examines the timing of the video signals in the SDI data streams and determines which video format is being received. The operation of this function is independent of and not dependent on the ST 352 payload ID packets. This function determines the transport format, not the picture format. Usually these are the same, but not always. For example, when 1080p 60 Hz video is transported on 3G-SDI level B-DL, the video transport is actually 1080i 60 Hz - the transport is interlaced, but the picture is progressive.

The transport format detection function makes it determination of the transport format purely by looking at the video timing. It cannot distinguish between video formats that have exactly the same timing. For example, progressive segmented frame (PsF) video formats are intentionally designed to with identical timing to corresponding interlaced formats and cannot be distinguished from interlaced formats by examining the timing. The transport format detection function reports PsF video formats as interlaced formats (rx_t_scan_out is Low). The user application must examine the ST 352 payload ID packets to discover whether the actual video format is PsF or interlaced.

6G-SDI and 12G-SDI mappings defined by SMPTE always subdivide the image into multiple sub-images. Each sub-image is formatted as a regular 1080p image. The transport format detection function examines the timing of data stream 1 (DS1) only. Thus, it reports the video transport of 6G-SDI and 12G-SDI signals as 1080p signals (rx_t_format_out = 0000 and rx_t_scan_out = 1) because this is what it detects on data stream 1.

The rx_t_family_out port provides a 4-bit code that indicates the matching video format family. The encoding of this output port is shown in Table 8. The transport detection function also determines whether or not the transport is interlaced or progressive and reports this on the rx_t_scan_out output port.

Table 8: **rx_t_family_out Encoding**

rx_t_family_out	Transport Video Format	Active Pixels
0000	SMPTE ST 274	1920 x 1080
0001	SMPTE ST 296	1280 x 720
0010	SMPTE ST 2048-2	2048 x 1080
0011	SMPTE ST 295	1920 x 1080
1000	NTSC	720 x 486
1001	PAL	720 x 486
1111	Unknown	
Others	Reserved	

The transport detector also determines the frame rate of the transport signal. The rx_t_rate_out port indicates the frame rate of the transport signal as shown in Table 9. The encoding of the frame rate matches the encoding used in the picture rate field of SMPTE ST 352 payload ID packets. However, rx_t_rate_out shows the transport frame rate, not the picture rate. Also, the rx_t_rate_out port value is always the frame rate, even for interlaced transports.

Table 9: **rx_t_rate_out Encoding**

rx_t_rate_out	Frame Rate
0000	None
0010	23.98 Hz
0011	24 Hz
0100	47.95 Hz
0101	25 Hz
0110	29.97 Hz
0111	30 Hz
1000	48 Hz
1001	50 Hz
1010	59.94 Hz
1011	60 Hz
Others	Reserved

UHD-SDI Wrapper Parameters

The UHD-SDI wrapper has three parameters. They are:

- FXDCLK_FREQ: This integer parameter specifies the frequency of the clock connected to the wrapper's clk_in port in Hz. The default value is 27000000 (27 MHz). The clk_in clock is used

for timing purposes in the UHD-SDI wrapper and also as a reference frequency to compare against the GTX RX recovered clock to determine the RX bit rate. If the FXDCLK_FREQ parameter does not match the frequency of the clock on clk_in, then these functions are compromised.

- DRPCLK_PERIOD: This integer parameter specifies the period of the clock connected to the wrapper's drpclk_in port in ns. The actual period of the clock must be rounded down to the next lowest integer. The drpclk_in clock is used for to generate some timing delays, such as the timeout periods of the GTX resets.
- INCLUDE_RX_EDH_PROCESSOR: This parameter must be specified as a string, either "TRUE" or "FALSE". If this parameter is "TRUE", the EDH processor is included in the UHD-SDI RX core. If the parameter is "FALSE", the EDH processor in the RX section of the UHD-SDI core is omitted.

GTX Controller

The UHD-SDI wrapper contains control logic for the GTX. This control logic initializes the GTX after FPGA configuration, resets the GTX in response to reset inputs to the UHD-SDI wrapper, and dynamically changes the operating mode of the GTX in response to changing SDI modes of the RX and TX sections.

The control logic for the RX and TX portions of the GTX are separate so that the RX and TX and controlled independently. Both GTX RX and TX control logic modules are based on Xilinx PicoBlaze microcontrollers. These microcontrollers are programmed to implement the GTX initialization, reset, and dynamic mode change sequences. During these sequences, the microcontroller might encounter abnormal conditions. The abnormal conditions often manifest themselves by a signal not being asserted within the expected period of time. For example, after asserting the gttxreset, the microcontroller expects the GTX to negate its txresetdone and then assert it again within a certain period of time. Such timeouts cause the microcontroller to retry certain phases of the sequence some number of times. If the sequence still fails after repeated attempts, the microcontroller indicates that a failure has occurred.

Both the RX and TX microcontrollers have three ports used to monitor their status. These ports are listed in Table 10.

Table 10: **GTX Control Logic Status Ports**

	Port Name	Description
RX	rx_change_done_out	High when the most recent RX sequence completed successfully.
	rx_change_fail_out	High when the most recent RX sequence failed.
	rx_change_fail_code_out	When rx_change_fail_out is High, this port indicates the nature of the failure.
TX	tx_change_done_out	High when the most recent TX sequence completed successfully.
	tx_change_fail_out	High when the most recent TX sequence failed.
	tx_change_fail_code_out	When tx_change_fail_out is High, this port indicates the nature of the failure.

Under normal conditions, the rx/tx_change_done_out port is High. When a change sequence is underway, the rx/tx_change_done_out port goes Low as soon as the microcontroller begins a

reset or change sequence and it remains Low throughout the duration of the sequence. At the end of the sequence, the rx/tx_change_done_out port goes High if the sequence completed successfully. If there was a failure, the rx/tx_change_done_out port remains Low and the rx/tx_change_fail_out port goes High. The rx/tx_change_fail_out code changes values during the sequence, even to values not listed in the failure code table, but this does not indicate failures unless the rx/tx_change_fail_out port is High. The failure codes for both the RX and TX microcontrollers are the same and are listed in Table 11.

Table 11: **RX/TX GTX Controller Failure Codes**

Failure Code	Description
0	Reset timeout: This code indicates that the microcontroller timed out waiting for a reset done signal from the GTX after a reset was initiated.
1	DRP request timeout: The RX and TX microcontrollers share the GTX DRP. There is an arbiter that controls which one has access to the DRP. If a microcontroller requests the DRP from the arbiter, but doesn't receive a grant within a certain period of time, the sequence terminates with this error code.
2	DRP cycle timeout: When the microcontroller does a read or write cycle on the GTX DRP, it must see a DRPRDY signal from the GTX indicating completion of the DRP cycle. If the microcontroller times out waiting for the DRPRDY, it retries the DRP cycle several times. If the microcontroller continues to timeout waiting for DRPRDY on the retries, it eventually fails to generate this error code.
3	Illegal PLL output divider: This error code indicates that the microcontroller was commanded to change the PLL output divider to an unsupported value.

If one of the microcontrollers asserts its change_fail_out port, the associated portion of the GTX must be considered to be in an invalid state. It could still be in a working condition, but any further changes to the SDI mode of that portion of the GTX is ignored by the failed microcontroller. When in the failure state, the only request that the microcontroller responds to is a full reset request. So, for example, if the RX microcontroller experiences a sequence failure and asserts its rx_change_fail_out port, the application must assert the rx_gtx_full_reset_in port High to attempt to restore proper operation of the GTX RX. The microcontroller, in response to assertion of the rx_gtx_full_reset_in port, attempts to do a full reset of the GTX RX. If that reset is successful, the rx_change_done_out port is asserted High and the GTX RX is functional. If the reset sequence fails, the rx_change_fail_out port is asserted High and the GTX RX is still not operational.

SDI Timing Constraints

The UHD-SDI core must have proper timing constraints applied. In this beta release version, it is completely up to the designer to apply these constraints properly. The included example design does have a correct set of constraints applied and these constraints can be used as a guide for implementing the correct timing constraints.

The main clocks associated with the UHD-SDI core and wrapper must have clock period constraints applied. These clocks are the RXOUTCLK and TXOUTCLK of the GTX and the clocks applied to the UHD-SDI wrapper's clk_in and drpclk_in ports, which are usually driven by the same clock. In the example design, the file named kc705_uhdsdi_demo_timing_6G.xdc or kc705_uhdsdi_demo_timing_12G.xdc has the timing constraints for these clocks.

To constrain the GTX RXOUTCLK and TXOUTCLK, use constraints like the following two clock period constraints:

```
create_clock -period 3.367 -name tx0_outclk -waveform {0.000 1.684} [get_pins
SDI/GTX/gtxe2_i/TXOUTCLK]

create_clock -period 3.367 -name rx0_outclk -waveform {0.000 1.684} [get_pins
SDI/GTX/gtxe2_i/RXOUTCLK]
```

In these two constraints, the UHD-SDI wrapper has an instance name of SDI. So, the hierarchy path to the GTX instance in the UHD-SDI wrapper is SDI/GTX/gtxe2_i. Replace the SDI portion of hierarchy path with the instance name of the UHD-SDI wrapper in your application. For each UHD-SDI wrapper, create constraints for each TXOUTCLK and RXOUTCLK and give each clock a unique name such as tx0_outclk, tx1_outclk, etc.

In the example TXOUTCLK and RXOUTCLK constraints, these clocks are constrained to 297 MHz which is the correct frequency to use when supporting 12G-SDI. If the maximum line rate supported by an application is 6G-SDI or lower, then these clocks should be constrained to 148.5 MHz (period of 6.734).

Also apply clock period constraints to the clocks connected to the UHD-SDI wrapper's clk_in and drpclk_in ports.

A set_clock_groups -asynchronous constraint should be applied to these clocks and any other clocks in the design so that Vivado doesn't treat the clock as being related. In the example design, there are four clocks: the RXOUTCLK, the TXOUTCLK, CLKOUT0 from MMCM and a clock called mgtclk from which the clock applied to the drpclk_in and clk_in ports of the UHD-SDI wrapper is derived. The following set_clock_groups command is applied to these three clocks:

```
set_clock_groups -asynchronous -group tx0_outclk -group [get_clocks rx0_outclk
-include_generated_clocks] -group [get_clocks mgtclk -include_generated_clocks] -group
[get_clocks -of_objects [get_pins
i_system_basic/system_basic_i/clk_wiz_1/inst/mmcm_adv_inst/CLKOUT0]]
```

The rx0_outclk includes option for -include_generated_clocks. This is because of the constraint that is described next. Also, the mgtclk has the -include_generated_clocks option because the clock connected to the clk_in and drpclk_in ports of the UHD-SDI wrapper is a derived clock, half the frequency of mgtclk.

The UHD-SDI wrapper contains the NI-DRU used to recover data in SD-SDI mode. The NI-DRU only runs in SD-SDI mode and in that mode, the RXOUTCLK has a frequency of 148.5 MHz. In applications that support 12G-SDI, the RXOUTCLK is constrained to 297 MHz and the NI-DRU does not usually meet timing at 297 MHz. But, it doesn't need to because it is only active when RXOUTCLK is 148.5 MHz. An additional set of constraints can be applied to the NI-DRU so that the NI-DRU is constrained to 148.5 MHz while the rest of the RX section is constrained to 297 MHz. The following two constraints are used in the example design to accomplish this:

```
set_property KEEP_HIERARCHY true [get_cells SDI/GTXCTRL/NIDRU]

create_generated_clock -name nidru_clk -source [get_pins SDI/GTX/gtxe2_i/RXOUTCLK] \
-divide_by 2 [get_pins SDI/GTXCTRL/NIDRU/CLK]
```

A KEEP_HIERARCHY constraint is applied to the NI-DRU module so that the clock name that must be identified in the next constraint isn't changed by synthesis. The get_cells portion of this constraint uses a path to the NI-DRU of SDI/GTX/NIDRU. In the example design, the UHD-SDI wrapper has an instance name of SDI. Change the SDI portion of this path to the instance name of the UHD-SDI wrapper in your application. The KEEP_HIERARCHY constraint is only applied for synthesis and does not apply for implementation so it does not interfere with any optimizations that the implementation tool may perform.

The create_generated_clock constraint creates a hierarchical clock just for the NI-DRU. This is not a physically separate clock. It is a logical clock used for timing analysis only. The NI-DRU is still driven by the RXOUTCLK of the GTX. This constraint tells the timing analyzer that the clock connected to the NI-DRU's CLK port is derived from the GTX RXOUTCLK but is half the maximum frequency. RXOUTCLK is constrained to 297 MHz, so the NI-DRU is constrained to 148.5 MHz.

CAUTION! *Do not apply the create_generated_clock constraint to the NI-DRU if the RXOUTCLK has been constrained to 148.5 MHz. Doing so would constrain the NI-DRU to 74.25 MHz operation, when it actually must be constrained to 148.5 MHz. Only apply this constraint if RXOUTCLK has been constrained to 297 MHz.*

The example design also applies another constraint file containing timing constraints to the design. This file is named v_smpte_uhdsdi_timing_constraints.xdc. The timing constraints in this file apply multicycle path constraints to the RX and TX EDH processors. The EDH processors only run in SD-SDI mode. Furthermore, these processors use a clock enable from the NI-DRU which is never asserted more often than once every five cycles of RXOUTCLK when RXOUTCLK it is running at 148.5 MHz. The EDH processors would normally default to the 297 MHz constraint applied to RXOUTCLK and they cannot meet that timing. So, four constraints are used to apply multicycle path constraints to the two EDH processors. There are two constraints for each processor. If the design is built without the optional RX EDH processor by setting the UHD-SDI wrapper's INCLUDE_RX_EDH_PROCESSOR parameter to "FALSE", then the two multicycle path constraints for the RX EDH processor should not be used.

The multicycle path constraints for TX EDH processor are shown next. One sets the setup time and the other the hold time.

The setup time is set to 10 clock cycles and the hold time is set to 9 clock cycles. This is the appropriate amount if RXOUTCLK is constrained to 297 MHz. However, if RXOUTCLK is constrained to 148.5 MHz, the setup time should be 5 clock cycle and the hold time should be 4 clock cycles.

```
set_multicycle_path -setup -from [get_cells * -hier -filter {name =~ *TX/TXEDH* && \
IS_PRIMITIVE && IS_SEQUENTIAL}] 10

set_multicycle_path -hold -from [get_cells * -hier -filter {name =~ *TX/TXEDH* && \
IS_PRIMITIVE && IS_SEQUENTIAL}] 9
```

The multicycle path constraints for the RX EDH processor are shown next. One sets the setup time to 10 clock cycle and the other the hold time to 9 clock cycles. As with the TX EDH processor, these are the correct values to use if TXOUTCLK is constrained to 297 MHz. However, if TXOUTCLK is constrained to 148.5 MHz, the setup time should be changed to 5 clock cycles

and the hold time to 4 clock cycles. These constraints should not be used if the RX EDH processor is not included in the design.

```
set_multicycle_path -setup -from [get_cells * -hier -filter {name =~ *RX/INCLUDE_EDH* \
&& IS_PRIMITIVE && IS_SEQUENTIAL}] 10

set_multicycle_path -hold -from [get_cells * -hier -filter {name =~ *RX/INCLUDE_EDH* \
&& IS_PRIMITIVE && IS_SEQUENTIAL}] 9
```

The multicycle path constraints use a wildcard at the beginning of the path. These constraints match any number of UHD-SDI wrappers located at any point in the hierarchy of the design. Therefore, if multiple UHD-SDI wrappers are used in a project, only one set of these constraints are required to properly constrain the EDH processors of all UHD-SDI wrappers. However, to prevent this file from being applied to other modules, it is best to set the SCOPED_TO_REF property of the v_smpte_uhdsdi_timing_constraints.xdc file to the v_smpte_uhdsdi_edh_processor module. This can be done in the Vivado GUI or by executing the following command from the Vivado Tcl Console:

```
set_property SCOPED_TO_REF {v_smpte_uhdsdi_edh_processor} [get_files \
v_smpte_uhdsdi_timing_constraints.xdc]
```

Example Design

This application note provides an example UHD-SDI design. This design runs on the KC705 FPGA evaluation board. It requires a Fidus 12G-SDI FMC board connected to the HPC FMC connector of the KC705 board. The example design has a single UHD-SDI transmitter driven by a test pattern generator. It supports operation at SD-SDI, HD-SDI, 3G-SDI (levels A and B), 6G-SDI, and 12G-SDI. The UHD-SDI transmitter is controlled by a Vivado Analzyer VIO module. The example design also has a single UHD-SDI receiver which can operate in the same modes as the transmitter. The status of the UHD-SDI receiver is monitored by a Vivado Analyzer VIO module. The data streams, line numbers, and video timing signals output by the UHD-SDI receiver are captured by a Vivado Analyzer ILA module and can be inspected in the Vivado Analyzer tool. Figure 15 is a block diagram of the example design.

A pre-generated FPGA configuration bit file is provided which can be loaded onto a KC705 board. However, 12G-SDI operation requires a -3 speed grade Kintex-7 FPGA. Standard KC705 boards have -2 speed grade devices and the bit file (probably) does not work on them at 12G-SDI rates. The bit file should work at 6G-SDI rates and lower on KC705 boards with -2 speed grade devices. However, this is not guaranteed because the bit file was generated for -3 devices.

The MGTAVCC voltage rail must be set to 1.05V if 12G-SDI operation is required. This voltage level supports the other SDI lines rates, too. If the maximum line rate is 6G-SDI or slower, then -1 speed grade devices are sufficient and the MGTAVCC voltage rail can be set to the normal value of 1.00V. Procedure details of setting up to adjust MGTAVCC voltage rail can be found from the link below. MGTAVCC is address 53, Rail #3. Also see Xilinx Answer Record 64007.

http://forums.xilinx.com/xlnx/attachments/xlnx/XLNXBRD/7895/1/KC705_Power_Controllers_Monitoring_Steps%20%281%29.pptx

All source code files for the example design are included. A Tcl script is provided that creates a Vivado project, add all of the source code files, and then implement the design and generate a bit file. The readme.txt file that accompanies the example design has instructions for using the Tcl script to generate the project.

This release was generated and tested with Vivado 2018.1. It does not work with earlier versions of Vivado. To control and monitor the example design, Vivado 2018.1 Analyzer is required.

This example design has limited support for 12G-SDI. Only one 12G-SDI line rate, either 11.88 Gb/s or 11.88/1.001 Gb/s, is available at a time. A DIP switch on the KC705 board determines whether the 12G-SDI line rate is 11.88 Gb/s or 11.88/1.001 Gb/s. This DIP switch controls the routing of the two reference clocks to the PLLs. When this DIP switch is closed, providing a High to the control logic, the QPLL is given the 148.35 MHz reference clock and the CPLL is given the 148.5 MHz reference clock. When the DIP switch is open, the reference clocks are reversed. So, if the DIP switch is closed, only the 11.88/1.001 Gb/s 12G-SDI line rate is supported. If the DIP switch is open, only the 11.88 Gb/s 12G-SDI line rate is supported.

The DIP switch should be set prior to configuring the FPGA. While the design continues to work if the DIP switch is changed while the FPGA is powered on and configured, the bouncing of the switch can cause issues, particularly for the RX section and proper operation is not guaranteed if the switch is changed dynamically.

The DIP switch that controls the reference clock routing is located in SW11, the GPIO DIP Switch. There are four DIP switches in this device and the switch labeled 4 is the switch that controls the reference clock routing. When this switch is in the down position, it is closed and only the 11.88/1.001 Gb/s 12G-SDI line rate is supported. Down, in this case, is towards the LCD display on the KC705 board.

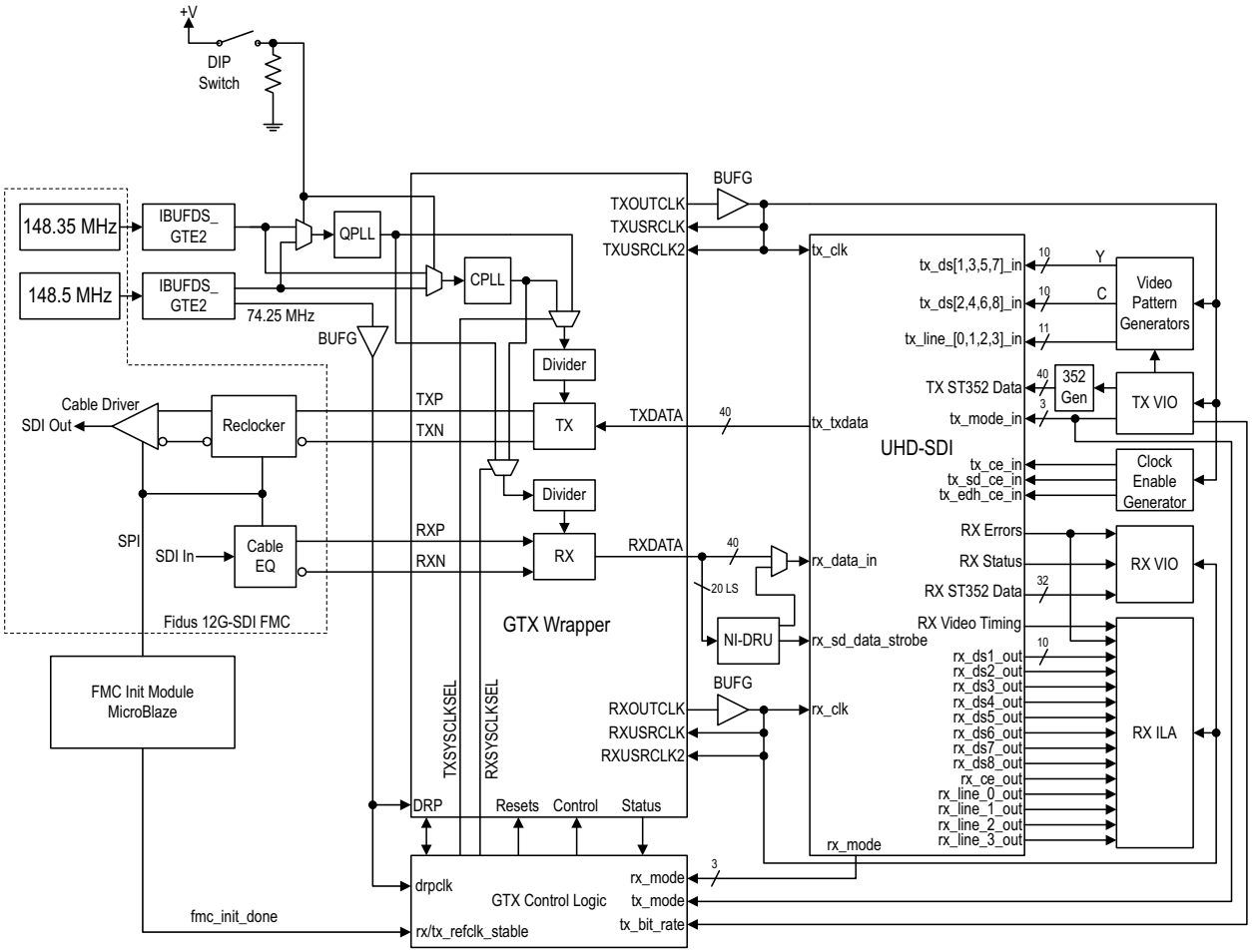

Figure 15: **UHD-SDI Example Design Block Diagram**

The transmitter is driven by a video pattern generator that can generate color bars or SDI pathological patterns in NTSC, PAL, 720p, 1080i, and 1080p formats at various frame rates.

When the transmitter is configured in 3G-SDI level B mode, the example design operates the transmitter in 3G-SDI level B-DS (dual stream) mode. In this configuration, a 1080p 25 Hz, 29.97 Hz, or 30 Hz image is generated by the video pattern generator and two copies of the that image are aggregated together and carried by the 3G-SDI level B-DS transport.

When the transmitter is configured in 6G-SDI mode, a 1080p 25 Hz, 29.97 Hz, or 30 Hz image is generated by the video pattern generator and four copies of that image (each copy consisting of a Y data stream and a C data stream) are sent to the data stream 1 through 8 inputs of the transmitter. This emulates what occurs when subdividing a 3840 x 2160 25, 29.97, or 30 Hz progressive 4:2:2 YCbCr 10-bit image per ST 2081-10 mapping mode 1 for transport on a 6G-SDI interface. Because the image is a color bar pattern which is highly symmetrical, the resulting image after the four 1080p sub-images are reintegrated into a 2160p image is a correct 2160p 25 Hz, 29.97 Hz, or 30 Hz color bar image.

When the transmitter is configured in 12G-SDI mode, a 1080p 50 Hz, 59.94 Hz, or 60 Hz image is generated by the video pattern generator and four copies of that image are sent to the data stream 1 through 8 inputs of the transmitter. This emulates what occurs when subdividing a 3840 x 2160 50, 59.94, or 60 Hz progressive 4:2:2 YCbCr 10-bit image per ST 2082-10 mapping

mode 1 for transport on a 12G-SDI interface. The resulting image after the four 1080p subimages are reintegrated into a 2160p image is a correct 2160p 50 Hz, 59.94 Hz, or 60 Hz color bar image.

A MicroBlaze processor configures the SDI PHY parts on the Fidus FMC board. After FPGA configuration, the MicroBlaze determines the die revision of the Macom SDI cable equalizer, reclocker, and cable driver and configures those devices appropriately. After configuration is complete, the MicroBlaze asserts a signal named fmc_init_done. This signal is connected to the UHD-SDI wrapper's rx_refclk_stable and tx_refclk_stable input ports so that the GTX and PLL resets do not complete until after the FMC is initialized and the reference clocks from the FMC card are stable. The fmc_init_done signal also drives an LED on the KC705 board. This LED is GPIO LED 7 located in the top right corner of the KC705 board, above and to the left of the power switch. If this LED is on, the FMC initialization is complete.

Running the Demo

To run this demo at 12G-SDI rates, a KC705 board with -3 speed grade XC7K325T FPGA in a FFG900 package is required. Connect a Fidus 12G-SDI FMC board to the HPC FMC connector as shown in Figure 16. Connect a USB cable from a USB port on the PC to the JTAG USB connector on the KC705. Connect the power cable to the KC705 board.

The instructions given here assume that the TX0 SDI output of the FMC card is looped back via a cable to the RX0 SDI input of the FMC card. Please keep in mind that the quality of that cable is critical at 12G-SDI line rates, even if it is very short. A poor quality cable significantly degrades the signal at 12G-SDI rates and prevents the UHD-SDI RX from properly receiving the signal.

The demo can also be used with TX0 driving a SDI sink, such as a SDI waveform monitor and RX0 connected to a SDI source. The RX and TX are completely independent.

Select the reference clock configuration using the Demo Configuration DIP switch. This is switch 4 on the Quad DIP switch shown in Figure 16. As previously described, this switch selects which line rate the RX and TX operate in when running in 12G-SDI mode.

Power on the KC705 board.

Demo Status LED

GPIO_LED_0- RX locked to SD-SDI mode

GPIO_LED_1- RX locked to HD-SDI mode

GPIO_LED_2- RX locked to 3G-SDI mode

GPIO_LED_3- RX locked to 6G-SDI mode

GPIO_LED_4- RX locked to 12G-SDI mode

GPIO_LED_5- RX bitrate indicator

GPIO_LED_6- RX change done indicator

GPIO_LED_7- FMC Initialization Done

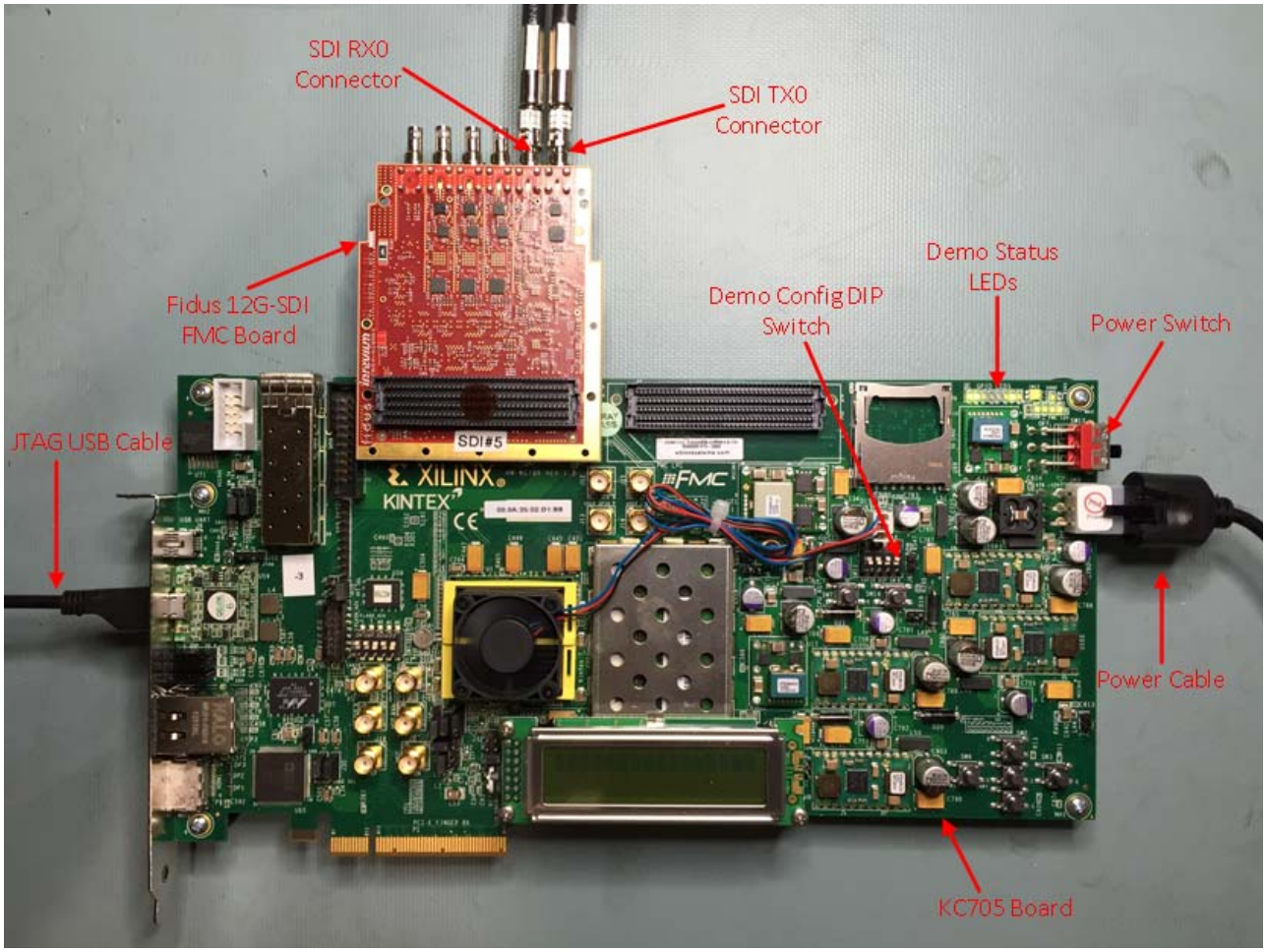

Figure 16: **KC705 and 12G-SDI FMC**

Open Vivado 2018.1.

In the Tcl Console, use the cd command to change the directory to the bit_files folder of the example design.

In the Tcl Console type: source bit_files.tcl. This is a Tcl script that opens a Vivado project called bit_files, opens the Vivado hardware manager, opens the target FPGA on the KC705 board, and programs the FPGA. The bit_files Vivado project is not a full project for the design. It only contains the setup information for the VIO and ILA modules used to control the demo. By using this project, you avoid having to configure how all of the IO signals are displayed.

After the Tcl scripts terminates, the hardware manager is open and there are three debug windows selectable by tabs. These windows are:

- ILA - hw_ila_1: This is the RX ILA setup window.
- VIO - hw_vio_1: This is (usually) the TX VIO window
- VIO - hw_vio_2: This is (usually) the RX VIO window

Vivado is not always consistent in which order the VIO modules are assigned, so it is possible for hw_vio_1 to be the RX VIO window and hw_vio_2 to be the TX VIO window.

Select the RX VIO window. Most of the signal names in the RX window start with rx0_. Not all of the signals are visible when you first open this VIO window. If your screen is large enough, you can pull down on the signal list resize control in the RX VIO window to make all of the signals visible as shown in Figure 17.

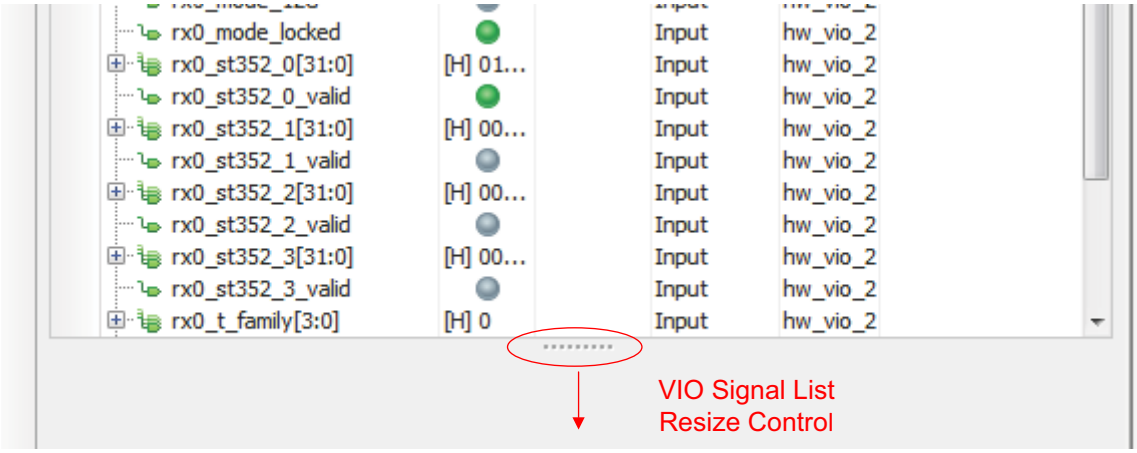

Figure 17: **RX VIO Signal List Resizing**

When the TX is looped back to the RX, it is usually more convenient to be able to see the RX VIO and the TX VIO at the same time. One way to do this is to float both of these windows by clicking on the float control in the upper right hand corner of the RX VIO window and the TX VIO window and then arranging these two floating windows side-by-side.

The TX VIO window controls the UHD-SDI transmitter. It is used to set the SDI mode and line rate, video pattern, video format, and select other options. The TX VIO window is shown Figure 18. The order of the signals may differ from what is shown in the figure.

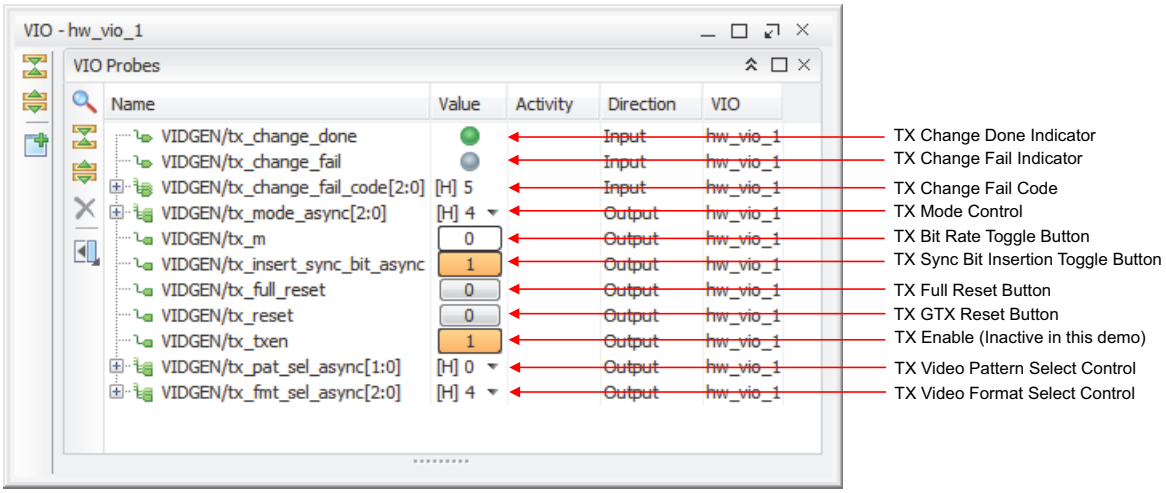

Figure 18: **TX VIO Window**

The first three items at the top of the TX VIO window indicate the status of the last GTX TX initialization or dynamic change sequence. If the last sequence completed normally, the

tx_change_done indicator is green. If the last sequence failed, the tx_change_fail indicator is red and the change failure code value indicates the cause of the failure as shown in Table 11.

The tx_mode_async value specifies the SDI mode and the tx_m toggle button selects between integer frame rates and fractional frame rates if appropriate for the selected SDI mode as shown in Table 12. The tx_m toggle button has no affect in SD-SDI and 12G-SDI modes. In SD-SDI, the only legal rate is 270 Mb/s. And, in 12G-SDI mode, the TX bit rate is controlled by the demo configuration DIP switch.

Table 12: **TX VIO SDI Mode and Line Rate Controls**

tx_mode_async	tx_m	Mode	Line Rate
0	0	HD-SDI	1.485 Gb/s
	1		1.485/1.001 Gb/s
1	NA	SD-SDI	270 Mb/s
2	0	3G-SDI Level A	2.97 Gb/s
	1		2.97/1.001 Gb/s
3	0	3G-SDI Level B-DS	2.97 Gb/s
	1		2.97/1.001 Gb/s
4	0	6G-SDI	5.94 Gb/s
	1		5.94/1.001 Gb/s
5	NA	12G-SDI	11.88 or 11.88/1.001 Gb/s

When tx_mode_async is set to 3, the transmitter is configured for 3G-SDI level B-DS operation. However, a value of 3 is illegal on the UHD-SDI wrapper's tx_mode_in port. The demo code uses tx_mode_async of 3 to indicate that the transmitter should run in 3G-SDI level B-DS mode. But, the logic changes this value to 2 on the tx_mode_in port of the UHD-SDI wrapper. On the UHD-SDI wrapper input, 3G-SDI level A or level B is selected by the tx_mux_pattern input port.

Immediately after the FPGA is programmed, the VIO windows do not necessarily represent the current status of the signals shown in the VIO window. This is particularly noticeable with the tx_mode_async and tx_m controls. The tx_mode_async and tx_m controls are displayed in the VIO window with whatever their previous value was. However, the actual tx_mode_async and tx_m signal values are returned to their default values in the FPGA. The default value of tx_mode_async is 0, selecting HD-SDI mode. And the default value of tx_m is 0, selecting 1.485 Gb/s line rate. The values in either VIO window can be updated by right clicking on the desired VIO module in the list of debug modules in the Hardware window (usually located in the upper left corner of the Hardware manager window) and selecting Refresh Input and Output Values for VIO Core. Whenever you have the situation where the TX is transmitting HD-SDI, but the tx_mode_async is set to some other SDI mode, the TX VIO has not been refreshed with the latest values.

The ST 2081-1 (6G-SDI electrical standard) and ST 2082-1 (12G-SDI electrical standard) documents mandate a feature called sync bit insertion, sometimes known a run length mitigation or anti pothole. The feature breaks up the long runs of 1's or 0's into the SDI encoder during EAV, SAV, ADF sequences. Some early 6G-SDI and 12G-SDI implementations do not support this feature and are not able to receive a signal that was transmitted with this feature. The tx_sync_bit_async toggle button enables and disables sync bit insertion in the transmitter.

The TX VIO provides two reset buttons for the transmitter. The tx_full_reset button resets the CPLL, the GTX TX, and the UHD-SDI TX data path. The tx_reset button resets the GTX TX and the UHD-SDI TX data path, but not the CPLL. When demo is on loopback, an rx0_manual_gtrxreset must be performed after a tx_full_reset to resume normal operation. This is because TX and RX are using the same PLL when both are in the same video format hence performing a tx_full_reset affects the RX operation.

The tx_txen control is not active in this demo.

The tx_pat_sel_async value selects the video pattern generated by the video pattern generator driving the SDI TX. In SD-SDI mode, two test patterns are available:

- 0 and 2 = SMPTE EG-1 color bars
- 1 and 3 = SMPTE pathological checkfield

In all other SDI modes, three test patterns are available:

- 0 = SMPTE RP 219 color bars
- 1 and 3 = SDI pathological checkfield
- 2 = 75% color bars

The pathological pattern produces the correct pathological checkfield pattern only in SD-SDI, HD-SDI, and 3G-SDI level A modes.

The tx_fmt_sel_async control selects the video format generated by the video pattern generator as shown in Table 13. Blank entries in the table are not supported, although the test pattern generator defaults to a legal pattern if unsupported combination is chosen.

Table 13: **TX VIO Video Format Selection**

tx_fmt_sel	SD-SDI	HD-SDI		3G-SDI Level A		3G-SDI Level B		6G-SDI		12G-SDI	
		tx_m=0	tx_m=1	tx_m=0	tx_m=1	tx_m=0	tx_m=1	tx_m=0	tx_m=1	DIPSW=0	DIPSW=1
0	NTSC	720p 50Hz									
1	PAL	1080pSF 24Hz	1080pSF 23.98Hz								
2	NTSC	1080i 60Hz	1080i 59.94Hz								
3	PAL	1080i 50Hz									
4	NTSC	1080p 30Hz	1080p 29.97Hz	1080p 60Hz	1080p 59.94Hz	1080p 30Hz	1080p 29.97Hz	2160p 30Hz	2160p 29.97Hz	2160p 30Hz	2160p 29.97Hz
5	PAL	1080p 25Hz		1080p 50Hz		1080p 25Hz		2160p 25Hz		2160p 25Hz	
6	NTSC	1080p 24Hz	1080p 23.98Hz								
7	PAL	720p 60Hz	720p 59.94Hz								

The RX VIO window shows the status of the UHD-SDI RX as shown in Figure 19. The order of the signals may be slightly different than shown in the figure.

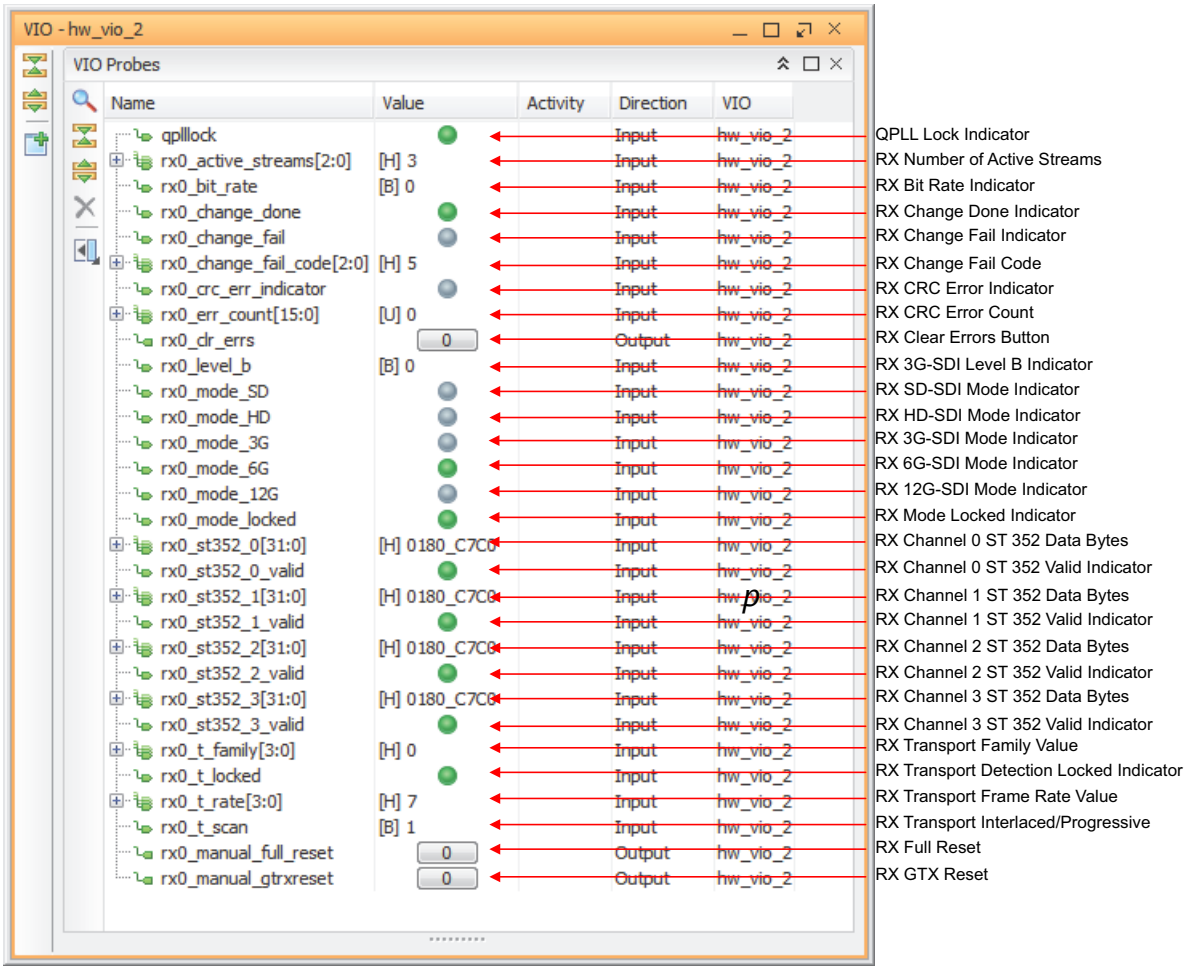

Figure 19: **RX VIO Window**

The QPLL lock indicator is green if the QPLL is locked to its reference clock. If this indicator is red, the QPLL is not locked, indicating a problem with the reference clock from the Fidus 12G-SDI FMC board.

The rx0_active_streams value indicates the number of active streams being received. This value is encoded as follows:

- 0 = 1 stream
- 1 = 2 streams
- 2 = 4 streams
- 3 = 8 streams
- 4 = 16 streams

The rx0_bit_rate indicator is 0 if an integer frame rate SDI signal is received and 1 if a fractional frame rate SDI signal is received.

The rx0_change_done indicator is green if the last initialization, reset, or dynamic SDI mode change completed successfully. If the sequence failed, rx0_change_fail is red and rx0_change_fail_code indicates the reason for the error as shown in Table 11.

The rx0_crc_error_indicator captures any occurrence of a CRC error in any active data stream. As soon as a single CRC error is seen, this indicator turns red. It stays red until cleared by clicking on the rx0_clr_errs button. In SD-SDI mode, per field EDH errors are captured by the rx0_crc_error indicator rather than per line CRC errors. The rx0_err_count value indicates the number of lines that contain a CRC error for all modes except SD-SDI and, in SD-SDI mode, the number of fields with an EDH error. The error counter saturates at 65535. The error counter is cleared by clicking the rx0_clr_errs button.

When receiving a 3G-SDI signal, the rx0_level_b indicator is green if the signal is level B and grey if it is level A.

The rx0_mode_SD, rx0_mode_HD, rx0_mode_3G, rx0_mode_6G, and rx0_mode_12G indicators show the SDI mode of the incoming signal. The indicator matching the mode of the SDI input signal is green. If none of the indicators are green, the receiver is not locked to the input SDI signal.

The rx0_mode_locked indicator is green if the UHD-SDI RX is receiving good data from the GTX RX.

There are four 32-bit values that display user data captured from ST 352 packets. Each ST 352 value also has a corresponding valid indicator which is green if ST 352 packets are being received for that channel. The order of the ST 352 user data byte in the 32-bit field is: {byte 4, byte 3, byte 2, and byte 1}. The value shown in rx0_st352_0 is from ST 352 packets received on data stream 1. The value shown in rx0_st352_1 is from ST 352 packets received on data stream 3, except in 3G-SDI level A mode when it shows ST 352 packets received on data stream 2. The value shown in rx0_st352_2 is from ST 352 packets received on data stream 5. And, the value shown in rx0_st352_3 is from ST 352 packets received on data stream 7.

The UHD-SDI RX has a transport format detector. This detector examines the timing of the SDI video transport and determines the format, frame rate, and scan mode (progress or interlaced) of the transport. This transport format is not necessarily the same as the picture format. For example, a 1080p 50Hz 4:2:2 10b image carried by 3G-SDI level B actually uses a 1080i 50Hz transport structure - the transport is interlaced but the picture is progressive. The rx0_t_locked indicator is green if the transport detector has identified the transport format. The rx0_t_family value indicates the format of the transport as listed in Table 8. The rx0_t_rate value indicates the frame rate (and always the frame rate even for interlaced transports) as listed in Table 9. And, the rx0_t_scan is 0 for an interlaced transport and 1 for a progressive transport.

The RX VIO window provides two reset buttons for the receiver. The rx0_manual_full_reset button resets the QPLL, the GTX RX, and the UHD-SDI RX data path. The rx0_manual_gtxreset button resets the GTX RX and the UHD-SDI RX data path, but not the QPLL. When demo is on loopback, a tx_reset must be performed after an rx0_manual_full_reset to resume normal operation. This is because TX and RX are using the same PLL when both are in the same video format hence performing an rx0_manual_full_reset affects the TX operation.

The ILA window captures the data streams, line numbers, and video timing signals output by the UHD-SDI receiver. Immediately after the hardware manager is opened, connected to the target, and the FPGA is programmed, there is one window for the ILA. This window is shown in Figure 20 and controls the setup of the ILA.

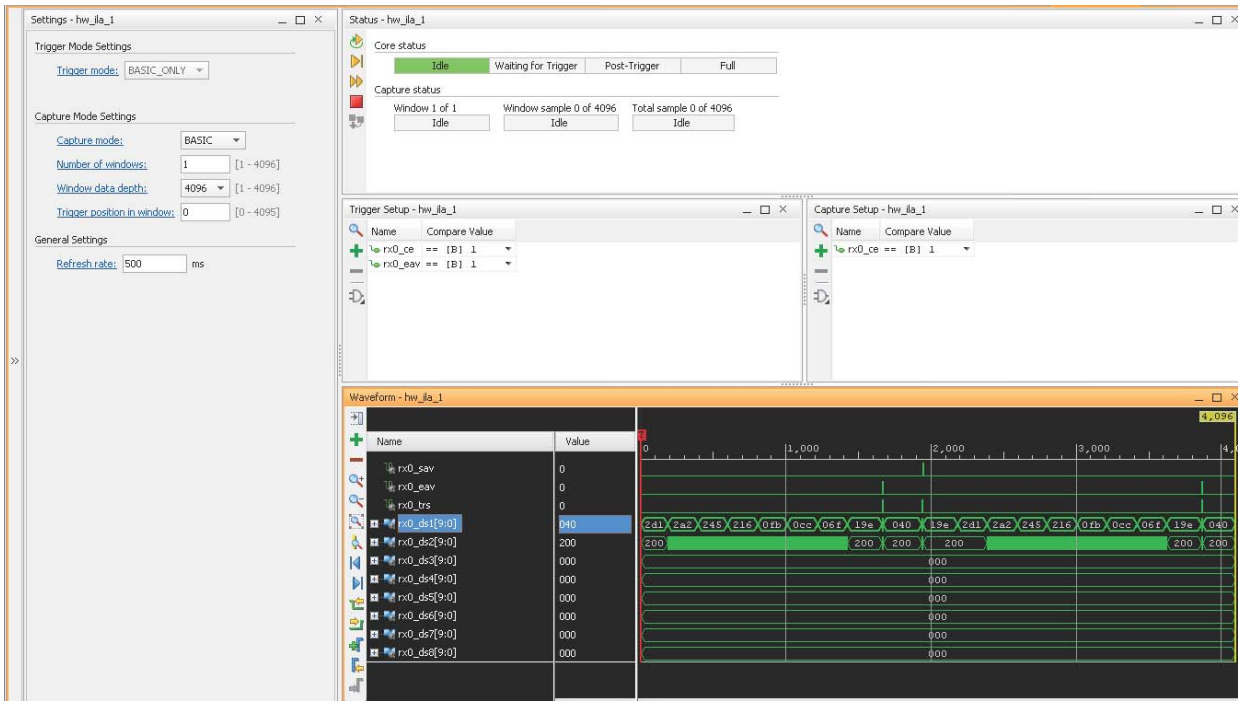

Figure 20: **RX ILA Setup Window**

By default, the RX ILA is setup to trigger on the last word of an EAV sequence. The ILA captures 20 samples before the trigger. The RX ILA is also setup to only capture samples when the rx0_ce clock enable is high. This allows about two lines of 1080-line video to be captured by the ILA. If you want to see how the clock enables work, the rx0_ce compare value in the Basic Capture Setup window can be changed to X, or the Capture mode can be changed to ALWAYS.

To capture data using the ILA, click on the trigger button, the button that looks like this:

After the ILA has captured the video and transferred it to Vivado Analyzer a new tab appears in Vivado Anzlyaer named hw_ila_data_1.wcfg. The contents of this new tab appears something like the waveform captured in Figure 21.

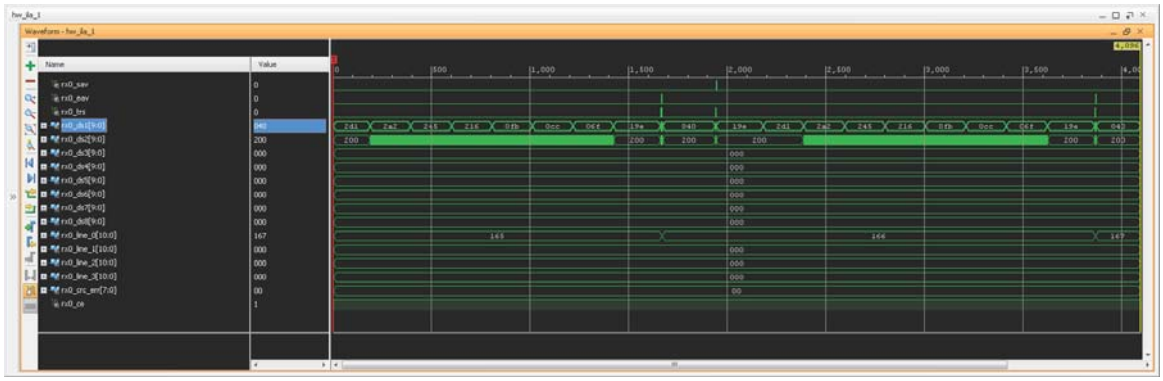

Figure 21: **RX ILA Capture Window**

This window shows the three video timing signals, rx0_eav, rx0_sav, and rx0_trs. It shows the data captured from the eight data streams output by the RX. One line number is shown for each pair of data streams. rx0_line_0 is the line number for data streams 1 and 2. rx0_line_1 is the line number for data streams 3 and 4, and so on.

The rx0_crc_err vector contains one CRC error bit for each of the eight data streams with bit 0 corresponding to data stream 1 and bit 7 corresponding to data stream 8.

Also shown is the clock enable, rx0_ce. If the capture mode is set so that data words are only captured when rx0_ce is High, then rx0_ce always appears to be High in the captured data.

inrevium 12G-SDI FMC Card Controller

This application note is supplied with IP Integrator based FMC controller to provide access and control to the I2C and SPI devices in the FMC card. The controller is instantiated in the project hierarchy as system_basic, and is composed of SPI, IIC GPIO IPs and basic IP components to run a simple MicroBlaze application. The GPIO's main purpose is to allow FMC channel selection during configuration and to indicate initialization done state.

There are three stages in the initialization process, first is the clock switch configuration which selects the two on-board crystal oscillator to supply 148.35 MHz and 148.5 MHz reference clocks to the GTX. Second is the generic SPI devices (cable driver, equalizer and reclocker) initialization, such as setting of output swing, input equalization factor and muting of one of two SDI cable driver outputs. Third is the SPI devices initialization based on the silicon version. Macom, the manufacturer of the cable equalizer and reclocker for the FMC card has published errata to their devices. Special SPI register initialization is needed for different silicon version (denoted by die ID) to ensure error free SDI reception. So far, errata 235x4-ERR-001-A, 23145-ERR-001-A, 23145-ERR-001-C, 23145-ERR-001-D and Errata M235x4-ERR-001-C were considered when the FMC controller was written.

There is a UART GUI interface to allow more flexibility on controlling the 12G-SDI FMC card. Users can selectively make register modifications to the I2C and SPI devices in the FMC card by following the instructions in the GUI shown below.

```
-----------------------
--  FIDUS Main Menu  --
-----------------------

 Select option
 1 = Re-Init
 2 = IIC Dev Select
 3 = SPI CH0 Select
 4 = SPI CH1 Select
 5 = SPI CH2 Select
 6 = SPI CH3 Select
 ? = help
------------------
>
```

Recompiling FMC Controller SDK Project

The SDK environment must be prepared after the completion of "kc705_uhdsdi_demo_script.tcl" script, this is done by exporting the project's hardware information and importing the SDK source codes.

1. Export hardware: In Vivado 2018.1 select, File->Export->Export Hardware
   a. In the Export Hardware pop-up window, Check Include bitstream option
   b. Set the Export to: field correspondingly

      ```
      <unzip_dir>\xapp1249\srcs\fidus_fmc_ctlr\SW
      ```

2. Launch Xilinx SDK 2018.1 select, File->Launch SDK
   a. Set both **Exported location** and **Workspace** fields to:

      ```
      <unzip_dir>\xapp1249\srcs\fidus_fmc_ctlr\SW
      ```

   b. Once in SDK, create a new Board Support Package; File->New->Board Support Package. Assign Project Name as **fidus_fmc_ctlr_bsp** then click **Finish**.
   c. In **Board Support Package Settings**, click **OK**.
3. Import SDK Sources: In SDK 2018.1 select, **File->Import**.
   a. In the **Import** pop-up window, select **General->Existing Projects into Workspace**.
   b. Click **Next**.
   c. Click on **Browse**... button, and make sure that it points to the corresponding folders.

      ```
      <unzip_dir>\xapp1249\srcs\fidus_fmc_ctlr\SW
      ```

   d. Click **OK**.
   e. Make sure fidus_fmc_ctlr is checked.
   f. Click **Finish**.
4. Assign fidus_fmc_ctlr_bsp to fidus_fmc_ctlr.
   a. In SDK, right click on fidus_fmc_ctlr folder.
   b. Click on **Change Referenced BSP**.
   c. Select fidus_fmc_ctlr_bsp and click **OK**.

FPGA Resource Usage

Table 14 shows the FPGA resources required for a UHD-SDI interface with a 7 series GTX transceiver. The resource usage includes all the modules required to implement the interface included in one SDI wrapper instance. Resource usage is shown for various common configurations.

The results shown were achieved with Vivado 2018.1.

The SDI receiver and transmitter interface designs do not use any MMCM clock managers.

Typically, one global or regional clock is required for each SDI TX and for each SDI RX. In addition, one fixed frequency global clock is required for timing purposes in the SDI wrapper. This fixed frequency clock is usually also used as the GTX DRP clock. Only one such fixed frequency global clock is required no matter how many SDI interfaces are implemented in the FPGA.

Table 14: **7 Series GTX SDI Interface FPGA Resource Usage**

UHD-SDI IP and Wrapper Configuration		FF	LUT	Memory LUT	BUFG
Max Line Rate	**UHD-SDI Core**				
3G-SDI	RX with EDH processor	10414	6761	164	1
	RX without EDH processor	9624	6260	163	1
6G-SDI	RX with EDH processor	11841	7213	164	1
	RX without EDH processor	11051	6650	163	1
12G-SDI 8 Data Streams	RX with EDH processor	11843	7625	164	1
	RX without EDH processor	11053	7066	163	1
12G-SDI 16 Data Streams	RX with EDH processor	11923	7622	164	1
	RX without EDH processor	11133	7059	163	1

Reference Design

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=390743, registration required.

Tool Flow and Verification

The following checklist indicates the tool flow and verification procedures used for the provided reference design.

Table 15: **Reference Design Checklist**

Parameter	Description
General	
Target devices	7 series FPGA with GTX transceivers
Source code provided	Yes
Source code format	Verilog
Design uses code/IP from existing Xilinx application note/reference design, IP catalog, or third party	Yes, IP cores from Vivado IP Catalog
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	None
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/version used	Vivado 2018.1
Implementation software tools/version used	Vivado 2018.1
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KC705

References

Available from the Society of Motion Picture and Television Engineers (www.smpte.org):

1. *RP 165: Error Detection Checkwords and Status Flags for Use in Bit-Serial Digital Interfaces for Television*
2. *RP 168: Definition of Vertical Switching Point for Synchronous Video Switching*
3. *ST 259: Television - SDTV Digital Signal/Data - Serial Digital Interface*
4. *ST 291-1: Television - Ancillary Data Packet and Space Formatting*
5. *ST 292-1: 1.5 Gb/s Signal/Data Serial Interface*
6. *ST 344: Television - 540 Mb/s Serial Digital Interface*
7. *ST 352: Payload Identifier Codes for Serial Digital Interfaces*
8. *ST 372: Dual Link 1.5 Gb/s Digital Interface for 1920x1080 and 2048 x 1080 Picture Formats*
9. *ST 424: Television - 3 Gb/s Signal/Data Serial Interface*
10. *ST 425-1: Source Image Format and Ancillary Data Mapping for the 3Gb/s Serial Interface*
11. *ST 2081-1: 6Gb/s Signal/Data Serial Interface - Electrical*
12. *ST 2082-1: 12Gb/s Signal/Data Serial Interface - Electrical*

Available from Xilinx (www.xilinx.com):

13. *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182)
14. *Virtex-7 T and XT FPGAs Data Sheet: DC and AC Switching Characteristics* (DS183)
15. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
16. *LogiCORE IP SMPTE UHD-SDI Product Guide* (PG205)
17. *Using Kintex-7 GTX Transceivers for SDI Interfaces* (XAPP592)
18. *Clock and Data Recovery Unit based on 20-Bit-Wide Oversampled Data* (XAPP1240)

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
04/01/2018	1.2	Updated for Vivado 2018.1 support. Updated Table 13.
08/14/2015	1.1	Updated Table 13.
04/01/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices