XAPP298 (v1.0) November 2, 2001

Serial Digital Interface (SDI) Video Encoder

Author: John F. Snow

Summary

The ANSI/SMPTE 259M-1997 standard specifies a serial digital interface (SDI) for digital video equipment operating at either the 525-line, 60 Hz video standard or the 625-line, 50 Hz video standard.[1] The SDI standard describes how to transport both composite and component digital video over standard video coax. SDI is widely accepted and often forms the video transportation "backbone" of television studios and broadcast centers.

This is one in a series of application notes describing SDI implementation in Xilinx FPGAs. Figure 1 is a block diagram showing correlation between the various application notes and the elements of the SDI link.

This application note focuses on the SDI encoder. The reference design includes several implementations of the SDI encoder optimized for use with the Virtex™-II FPGA series and other Xilinx FPGA families. Both serial (bit-rate) and parallel (word-rate) implementations of the SDI encoder are presented. Also included are examples illustrating using a Xilinx FPGA as an alternative to several commercially available SDI encoder devices, the Gennum GS9002 and the Cypress CY7C9235.

A test bench and several diagnostic modules are included for testing the SDI encoder modules described in this application note, and the SDI decoder modules described in XAPP288: Video Decoder[4].

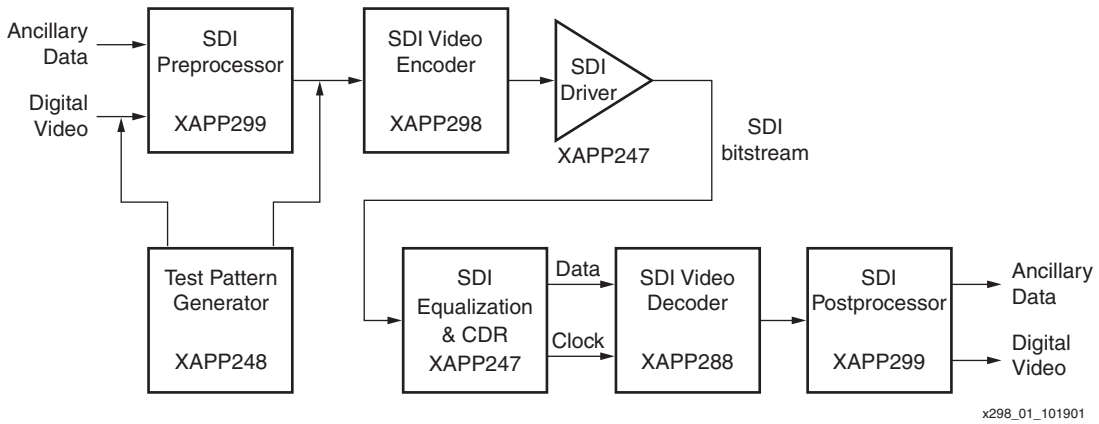

Figure 1: **SDI Block Diagram and Application Notes**

SDI Introduction

Digital Video Formats

The SDI standard describes how to transport standard definition digital video serially over a video coax cable. This standard describes the encoding and decoding processes performed on the video bitstream for transportation across the physical layer. The standard also describes the electrical and mechanical characteristics of the physical layer. However, it does not define the actual format of the digital video data. Additional standards for the definitions of SDI compatible digital video formats are:

- ANSI/SMPTE 125M, ANSI/SMPTE 267M, and ITU-R BT.601-5 for 4 x 3 and 16 x 9 aspect ratio 4:2:2 component digital video. [1][2]

- ANSI/SMPTE 244M for composite NTSC digital video [1]
- IEC 1179 (now called IEC 61179) for composite PAL digital video [3]

The SDI standard does not cover high definition digital video. Another standard, SMPTE 292M, defines a serial digital interface standard for high definition digital video, commonly called HD-SDI. The bandwidth requirements for high definition video are significantly higher than for standard definition video. Also, video components in the HD-SDI standard are interleaved differently than in the SDI standard. Because implementing an HD-SDI encoder involves higher bandwidth requirements and different formats than a standard definition SDI encoder, it is not covered in this application note.

All digital video formats supported by the SDI standard use either eight bit or ten bit-per-data word. Although the SDI standard always sends ten-bit data across the link, with proper handling it can transport eight-bit digital video formats. When eight-bit video is used, the two least significant bits of a ten-bit input to the SDI encoder may be tied High or Low.

Encoding and Decoding

Prior to sending digital video serially across the physical layer, an SDI transmitter must encode the video according to the SDI standard. By design, this encoding process ensures that the serial bitstream has sufficient level transitions to allow the receiver to recover the clock and data. After the receiver captures the serial data, the decoder must reverse the encoding process to recover the original video data.

The SDI standard uses two generator polynomials, normally expressed as linear feedback shift registers (LFSR), to implement two separate encoding stages. First, the video bitstream is scrambled using the generator polynomial:

$$G1(x) = x^9 + x^4 + 1$$

The output of this first encoding stage is referred to as the scrambled non-return-to-zero (NRZ) bitstream.

The second encoding stage uses the generator polynomial:

$$G2(x) = x + 1$$

It converts the scrambled NRZ bitstream to a polarity-free scrambled NRZ-inverted (NRZI) bitstream. NRZI is DC balanced for transmission across the physical layer. If the bitstream is inverted between the transmitter and the receiver, then the polarity-free nature of the SDI bitstream allows the decoder to properly recover the original data.

The SDI decoder reverses the encoding process by using the same generator polynomials in reverse order: G2 to convert from NRZI to NRZ and then G1 to descramble the bitstream.

Figure 2 illustrates the encoding and decoding processes when implemented in LFSRs. The circles with plus symbols inside are exclusive-OR gates. The boxes represent individual flip-flops. The LSB of a data word is sent first.

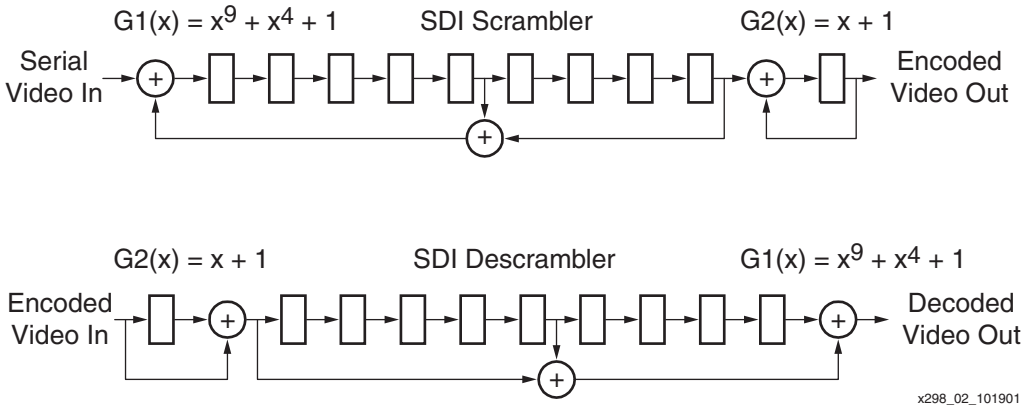

Figure 2: **SDI Encoding and Decoding Processes**

Framing and TRS Clipping

After decoding the video bitstream, the receiver must determine where individual ten-bit words begin and end in the serial bitstream. This process is called framing. In order to frame the bitstream, a unique and recognizable pattern must be sent periodically for the framer to use as a framing reference.

All of the digital video formats supported by SDI share similar definitions for the timing reference signal (TRS) symbols. TRS symbols delineate between the active and inactive portions of the video. For component video standards, two TRS symbols are sent per line of video: one at the start of the active video called SAV, and one at the end of active video called EAV. For composite video standards, one TRS symbol is sent per line. A TRS symbol is sent as four consecutive words, formatted as:

```
3ff  000  000  XYZ
```

The first three words of the TRS symbol, called the preamble, form a unique sequence in the bitstream. The fourth word, called XYZ, varies depending on the specific digital video format being transported.

Since the TRS preamble is common across all the supported digital video formats, is sent on a regular basis, and is unique in the bitstream, it is used as the framing reference. Upon detecting a sequence of ten consecutive ones and twenty consecutive zeros, a framer in the SDI receiver can determine the proper boundaries of all subsequent data words in the bitstream.

When transporting eight-bit digital video formats, the SDI transmitter must convert the eight-bit video words into the native SDI ten-bit format. This must be done properly in order to generate valid 10-bit TRS preambles. The transformation of eight-bit digital video into SDI compatible ten-bit digital video is called TRS clipping. The SDI standard requires forcing all data values between hex `3fc` and `3ff` to a value of `3ff` prior to encoding. Likewise, values between `000` and `003` must be forced to a value of `000`.

SDI Bit Rates

The bit rates supported by SDI range from 143 Mb/s to 360 Mb/s, depending on the digital video format being transported. The SDI standard defines four different bit rates as "support levels" (shown in Table 1). SDI compliant equipment is not required to support all bit rates. A piece of equipment supporting bit rates up to 270 Mb/s is said to conform to ANS/SMPTE 259M-ABC, since it supports levels A, B, and C.

Table 1: **SDI Standard Bit Rates**

Support Level	Bit Rate	Video Format	Standard
Level A	143 Mb/s	NTSC composite	ANSI/SMPTE 244M-1995
Level B	177 Mb/s	PAL composite	IEC 61179
Level C	270 Mb/s	4 x 3 4:2:2 component	ANSI/SMPTE 125M-1995 and ITU-R BT.601-5
Level D	360 Mb/s	16 x 9 4:2:2 component	ANSI/SMPTE 267M-1995 and ITU-R BT.601-5

Error Detection

The SDI standard does not mandate the use of an error detection mechanism. Some of the digital video standards, SMPTE 125M for example, specify error detection bits in the XYZ word of the TRS to determine the validity of the TRS symbol. However, the SDI standard highly recommends embedding error detection check words into the SDI video stream as described in SMPTE RP 165-1994. Techniques for generating and inserting these check words are described in XAPP299: Ancillary Data and EDH Processors. [5]

Clock Jitter Considerations

Any SDI encoder that generates a serial SDI bit stream generally requires a clock running at the SDI bit rate for the parallel-to-serial converter. A bit-rate serial clock can be generated in the FPGA by multiplying the parallel data clock using a Virtex-II DCM. Alternatively, an external clock multiplier circuit may be used to provide a serial clock to the FPGA.

The SDI standard allows for a maximum peak-to-peak jitter of 0.2 times the serial clock period. If the SDI link is running at 360 Mb/s, then the maximum jitter allowed is about 550 ps.

Parallel digital video standards allow relatively large amounts of clock jitter on the parallel clock. For example, the SMPTE 125M standard allows up to 3 ns of peak-to-peak jitter on the parallel clock. This can make the parallel clock unsuitable for use as a reference to the clock multiplier.

The Virtex-II DCM does not filter out clock jitter on the reference clock. Any jitter on the parallel clock becomes jitter on the serial clock, with additional jitter added by the DCM and the clock distribution network. If a parallel clock from a SMPTE 125M video source is used as the reference clock, the resulting serial clock jitter could greatly exceed the SDI jitter specification.

If the designer cannot ensure that the parallel clock has sufficiently low jitter to make it suitable for use as a reference to the DCM, then an external clock regenerator or clock multiplier capable of reducing the parallel clock jitter must be used.

Jitter considerations for SDI implementations in Xilinx FPGAs will be covered in more detail in XAPP247: MicroBlaze and Multimedia Development Board: SDI Physical Layer Implementation.

Reference Design

The reference design files are available on the Xilinx FTP site at: **XAPP298.zip**. The reference design includes several different SDI encoder implementations, a TRS clipper module, diagnostic modules, and a test bench.

TRS Clipper

TRS clipping is required to support eight-bit digital video in the ten-bit SDI protocol. The *trs_clipper* module is a simple combinatorial design. If the eight most significant bits of the video word are all zeros, the module forces the two least significant bits to zeros. If the eight most significant bits are all ones, the two least significant bits are forced to ones. Otherwise, the two least significant bits pass through the module unchanged. An enable input to the TRS clipper module is provided to disable the TRS clipping function if desired.

Figure 3 shows a block diagram of how the TRS clipper is used in an SDI transmitter. This block diagram uses a DCM to multiply the parallel clock by five. The serializer shifts out two bits every five clock cycles into DDR flip-flops. This was done instead of using the DCM to multiply the parallel clock by ten because the CLKFX output of the Virtex-II DCM currently can not run fast enough to generate a bit-rate clock for the highest SDI bit rates. By using a half bit-rate clock and the DDR hardware in the Virtex-II IOBs, a Virtex-II design can easily serialize data at the maximum SDI bit-rate.

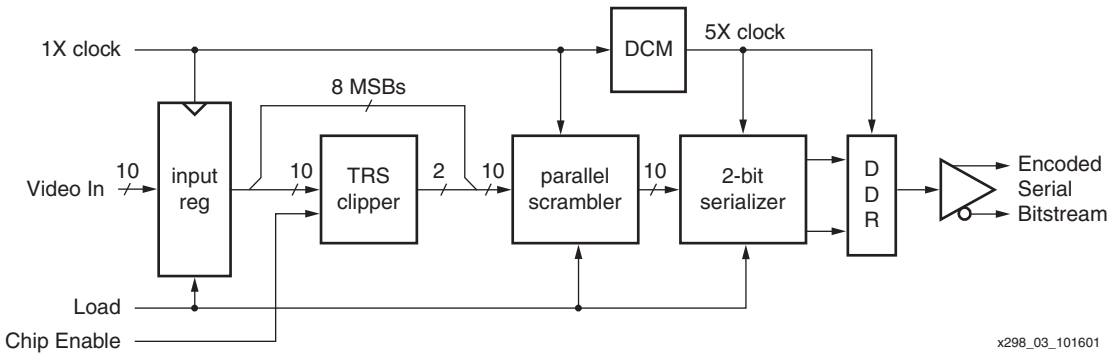

Figure 3: **Example SDI Transmitter with TRS Clipper**

Bit-Rate Serial Scrambler

The scrambling process involves "division" of the incoming bitstream by the generator polynomials. A simple LFSR implementation is shown in Figure 4. A serial implementation results in a very small amount of hardware. However, a serial implementation must run at the full bit-rate of the SDI interface, up to 360 MHz.

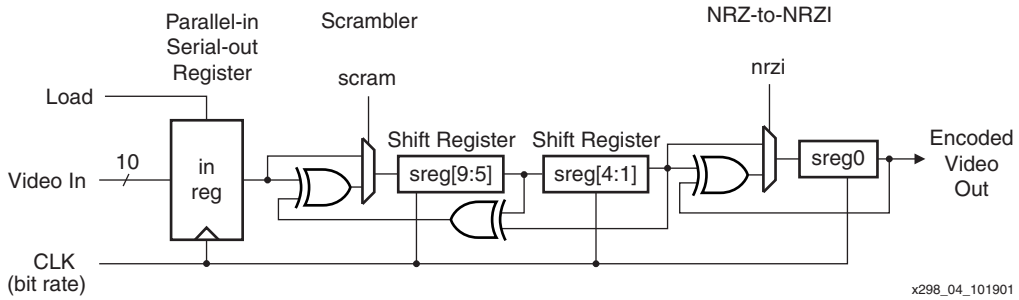

Figure 4: **Bit-rate Serial SDI Scrambler**

The HDL files *ser_scrambler.** contain the bit-rate serial SDI scrambler using a LFSR. As shown in **Reference Design Results**, this implementation is very small. In a Virtex-II FPGA, the serial scrambler runs fast enough to support the highest bit rate specified by the SDI standard.

The serial scrambler module has two control inputs, *scram* and *nrzi*, to enable the scrambler and the NRZ-to-NRZI conversion, respectively. These control signals allow the two encoding stages to be bypassed if the data to be sent is non-SDI compliant. In normal SDI operation, both inputs should be High.

Half Bit-Rate Serial Scrambler

The Virtex-II architecture features double data-rate (DDR) output flip-flops and a DDR MUX in the IOB. A scrambler module that processes two bits per clock cycle can be used to drive the DDR flip-flops in an IOB. This allows the clock to the scrambler to run at one-half the SDI bit rate rather than at the full bit rate as required for a serial SDI scrambler.

Figure 5 shows a block diagram of an SDI scrambler that processes two bits per clock cycle. The HDL files *ser_scrambler2.** contain the module shown in the block diagram. As the **Reference Design Results** section shows, this implementation is only slightly larger than the bit-rate serial scrambler previously described. Since it only needs to run at half the SDI bit-rate, this design can easily support the highest SDI bit-rates.

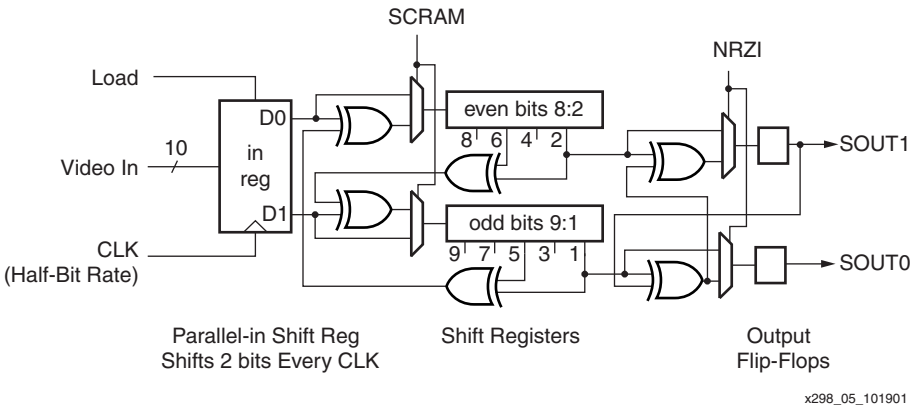

Figure 5: **SDI Scrambler Processing Two Bits Per Clock Cycle**

The *X9002* module described later in this application note is an example of how to use the *ser_scrambler2* module with the Virtex-II DCM and DDR features to implement an SDI encoder.

Parallel Scrambler

The scrambler function can also be implemented in a parallel manner, processing one ten-bit word every clock cycle. This requires more hardware but only needs to run at one-tenth the bit rate of the SDI link.

In some situations, it is advantageous to use a larger parallel scrambler implementation. Since the parallel descrambler only has to run at the word rate, lower performance FPGAs can be used to support the highest SDI bit rates. With some lower performance FPGAs, it may be necessary to use an external device to serialize and transmit the encoded parallel data generated by the FPGA.

Figure 6 shows a block diagram of the module described in the *par_scrambler.** files. This module accepts a ten-bit input word and generates a ten-bit output word every clock cycle. There are two clock cycles of latency through the scrambler. Also refer to Figure 3 for an example of using the *par_scrambler* in an SDI transmitter.

Ten 3-input XOR gates implement the SDI scrambler function. The incoming data bits are combined with the nine bits scrambled in the previous clock cycle and stored in the *scram_reg*. The least significant five bits of the incoming data word are scrambled and fed back into the scrambler to generate the five most significant data bits.

The NRZ-to-NRZI converter is implemented with ten two-input gates that XOR each bit with the bit that preceded it in the bitstream. This requires 11 bits to generate ten NRZI bits. The eleventh bit is the MSB stored in the *out_reg*.

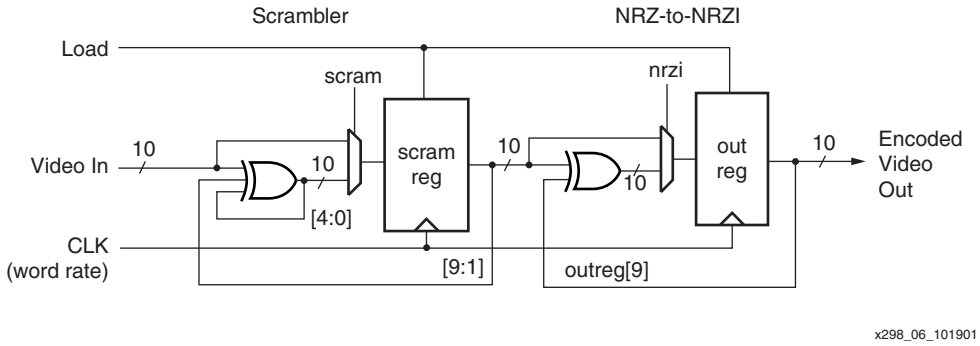

Figure 6: **Parallel Scrambler Block Diagram**

X9002 Example: An Alternative Solution

The Gennum GS9002 was one of the first commercially available SDI encoder integrated circuits. It contained a TRS clipper, an SDI scrambler, and a PLL used to generate the serial clock from the parallel data clock. Although the GS9002 is now obsolete, Xilinx FPGAs can provide an alternative solution when redesigning equipment originally using a GS9002.

Figure 7 is a block diagram of the *X9002* module provided in the reference design. A Virtex-II DCM multiplies the parallel data clock by five to synthesize a clock that runs at half the SDI bit rate. Two phases of this five times clock, 180° out of phase, are synthesized to drive the DDR logic. A TRS clipper circuit feeds clipped parallel video data to a *ser_scrambler2* module where it is encoded and serialized. Virtex-II DDR flip-flops and a DDR MUX are used to generate the SDI serial bitstream output.

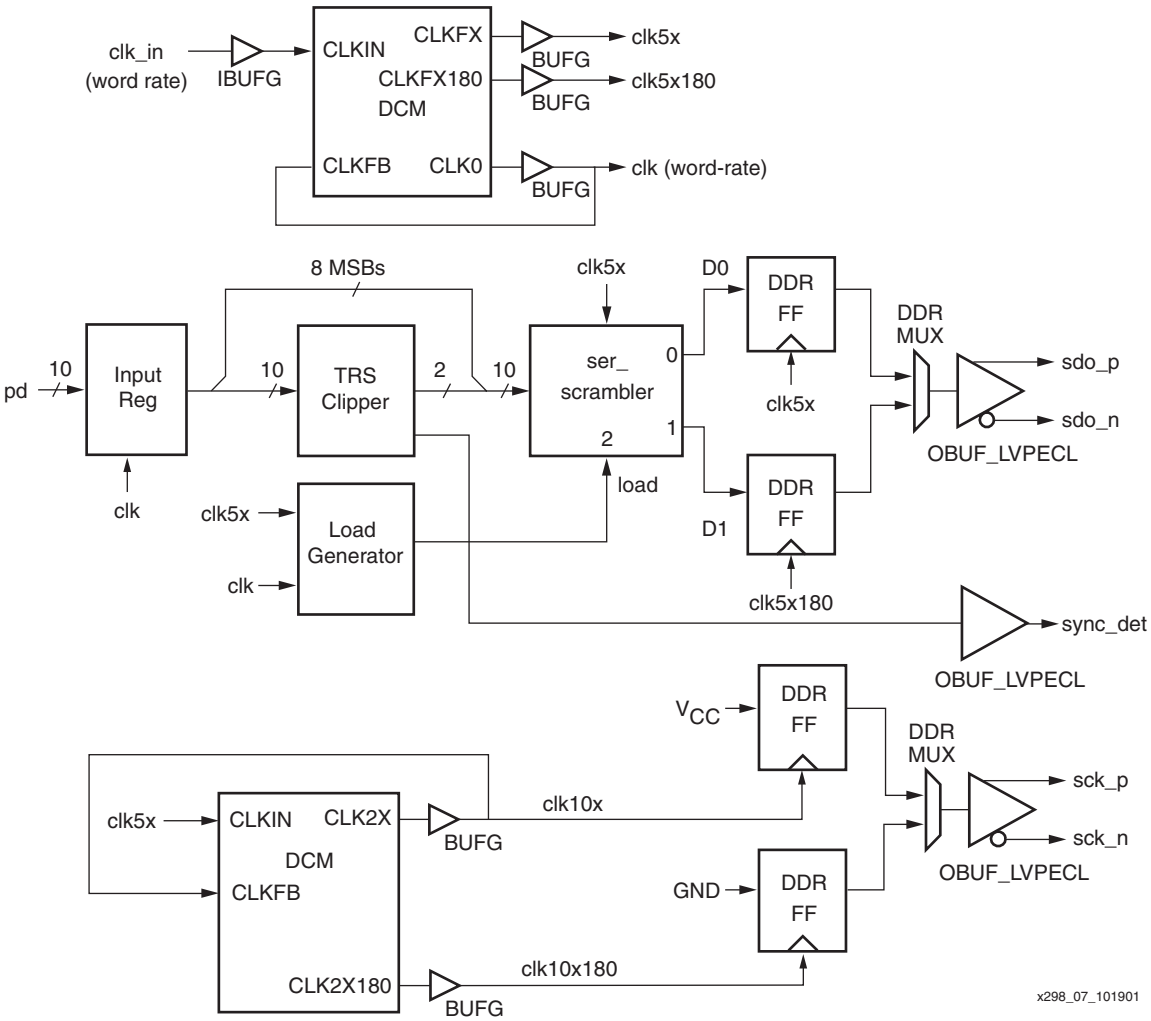

Figure 7: **X9002 Example Block Diagram**

A GS9002 provides a bit-rate clock through a differential PECL driver. This bit-rate clock is for reference and diagnostic purposes and is not required by the SDI standard. The *X9002* module duplicates this functionality by using a second DCM to double the five times clock. Rather than using the second DCM to multiply the parallel clock by a factor of ten, the second DCM doubles the five times clock generated by the first DCM. This cascaded DCM configuration was used because the CLKFX output of the Virtex-II DCM currently is not capable of generating a 360 MHz output, but the CLK2X output is capable of this speed.

DDR flip-flops are used to drive the serial clock out of the FPGA. This technique ensures a nearly 50% duty cycle on the clock output. This second DCM and associated global clock buffers and the DDR hardware can be easily removed from the design if the reference serial clock output is not required.

X7C9235 Example: An Alternative Solution

The Cypress CY7C9235 SMPTE 259M/DVB-ASI Scrambler-Controller is a parallel implementation of an SDI encoder. It accepts a ten-bit video word and generates a ten-bit encoded video word every clock cycle. The CY7C9235 is designed to operate in two modes, an SDI compliant mode and a DVB-ASI mode when used in conjunction with a Cypress CY7B9234 transmitter. In SDI mode, the CY7B9234 simply serializes the SDI encoded data supplied by the CY7C9235. In DVB-ASI mode, the CY7C9235 passes the data to the CY7B9234 unmodified and the CY7B9234 performs 8B/10B encoding on the data before serializing it.

Figure 8 is a block diagram of the *X7C9235* module provided in the reference design. The module includes an instance of the *par_scrambler* module to do parallel encoding of the video data. This design example does not implement the DVB-ASI mode features of the Cypress CY7C9235.

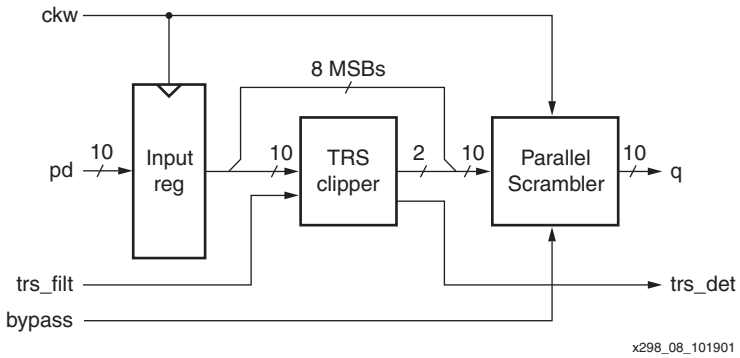

Figure 8: **X7C9235 Example Block Diagram**

Testing

Figure 9 shows the block diagram of a test bench developed for simulation verification of the SDI encoder modules in this application note and the SDI decoder modules from XAPP288: Video Decoder.[4]

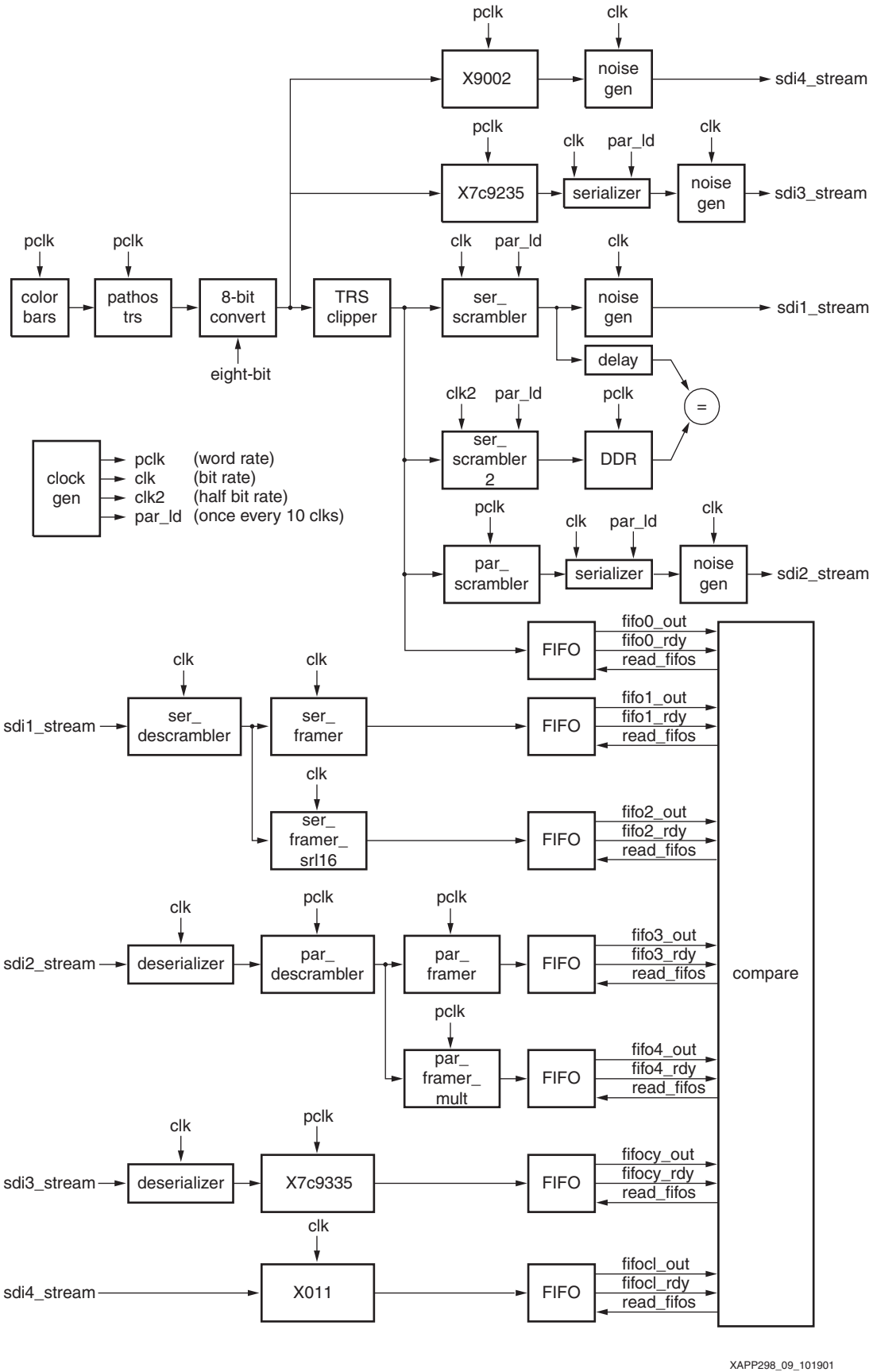

Figure 9: **SDI Test-Bench Block Diagram**

The test bench includes a simple color bar test pattern generator to serve as the source of the video to be sent through the SDI link. The color bar generator and other test pattern generators are described in XAPP248: Digital Video Test Pattern Generator.[7]

The video data generated by the color bar generator passes through a pathological TRS test case generator. This module passes the active video data and TRS symbols unchanged. During the horizontal blanking period, the module inserts pathological test sequences. These sequences resemble TRS symbols but differ by just a single bit.

The ten-bit digital video out of the pathological test case generator can be forced to a simulated worst case eight-bit value to exercise the TRS clipper. If the *eight_bit* signal is asserted, bit zero of the video is forced to a one and bit one is forced to a zero. This will cause the TRS clipper to clip both the all-zeros and all-ones cases. When the pathological test case generator is actively inserting its test cases, as indicated by the assertion of the *replace* signal, the TRS clipper must be disabled. If it were not, the TRS clipper could turn some of the TRS-like test cases into valid TRS symbols.

The output from the TRS clipper is fed into the various SDI scrambler modules and to a FIFO module. The data in the FIFO module is used as a reference for comparison against the data recovered by the SDI decoders.

The two-bit wide data path from the half bit-rate scrambler, *ser_scrambler2*, is connected to a DDR flip-flop pair and a MUX to generate a serial bitstream. This bitstream is directly compared against the serial bitstream from the serial scrambler, *ser_scrambler*, and any differences are reported as errors.

The bitstream from the ser_scrambler module passes through a noise generator module before passing to the serial decoders. This noise generator module is capable of corrupting the bitstream in two ways. The noise generator can inject a burst of noise that corrupts random bits in the bitstream. This is intended to simulate electrical noise injected onto the signal. This noise mode is not used in this testbench. It is intended for use with the error detection processors that will be described in XAPP299.

The noise generator can also insert or remove and random number of consecutive bits (from one to nine bits) from the bitstream. This will cause the data recovered by the SDI decoder to be unframed and invalid and will force the SDI decoder to reframe at the next TRS symbol. This simulates what happens if the SDI decoder becomes unsynchronized. In actual systems this can occur for a number of different reasons, such as when the video stream is switched to a different, unsynchronized video source. The noise generator only inserts or removes whole bits to test that the SDI decoder will detect the offset and reframe. It is not intended to simulate partial bit jitter for testing the clock and data recovery unit. The period between bit insertion or removal is controlled by the *OFF_PERIOD* parameter in the noise generator module code.

The parallel scrambler module also encodes the video data from the *trs_clipper* module. The data from the parallel scrambler is serialized and then sent through a noise generator module. The resulting bitstream is called *sdi2_stream* and drives the parallel SDI decoder chains.

The SDI bitstreams are each descrambled and framed by the descrambler and framer modules from XAPP288[4]. The data out of each framer is written into a FIFO module. The output of each FIFO module is compared against the reference video stream stored in the FIFO connected to the output of the *trs_clipper* module.

The *X9002* and *X7C9235* encoder modules contain their own TRS clipper circuits and are connected to the output of the 10-bit to 8-bit converter. There is a noise generator module connected to the output of each of these two modules. The serial bitstream from the *X7C9235* encoder drives the *X7C9335* decoder. The serial bitstream from the *X9002* encoder drives the *X011* decoder (see XAPP288[4]).

The *sdi_fifo* module is designed to simplify the comparison of video streams passing through SDI links that have different latencies. Framer designs react differently to the insertion of noise, especially offset noise. To make the comparison easier, the noise generators are monitored to determine when any of them inserts offset noise. As soon as this occurs, all the FIFOs are flushed and they stop storing data until the next TRS symbol is written into the FIFO. In this way,

only properly framed data from the various SDI decoder chains will be compared against the reference video data.

Verification of the actual hardware of an SDI link involves more than just verifying that a color bar pattern can be passed through the link. The primary areas of concern are the cable equalization and clock and data recovery. Refer to XAPP247[6] for more information about testing these areas.

The *patho_trs* module can be synthesized and placed in an SDI transmitter design. When enabled, the *patho_trs* module's pathological test cases are inserted during the horizontal blanking period. These test cases verify that the receiver's framer function does not falsely detect a TRS symbol when it receives bit sequences similar to TRS symbols. The *patho_trs* module also generates bit sequences that differ from ANC symbols by one bit. An ANC symbol represents the beginning of an ancillary data block. The ANC-like test patterns generated by *patho_trs* can be used to verify the proper operation of the receiver logic that looks for ANC blocks.

Reference Design Results

Table 2 shows the results after place and route of the various modules implemented in this application note. All results were obtained using the Verilog versions of the designs with Xilinx ISE version 4.1i. Results using the VHDL files are not shown but are essentially identical. Virtex-II results are for a -5 speed grade device. Spartan™-II results are for a -6 speed grade device.

The *ser_scrambler* module must run as fast as the bit rate of the SDI link. The *ser_scrambler2* and *X9002* modules run at half the bit rate. The *par_scrambler*, *trs_clipper*, *X7C9235*, and the *patho_trs* modules run at the word rate (one-tenth the bit rate).

Table 2: **Design Results**

File Name	XST		
	Size LUTs/FFs	Virtex-II Speed	Spartan-II Speed
ser_scrambler.v	12/20	450 MHz	280 MHz
ser_scrambler2.v	14/21	300 MHz	N/A
par_scrambler.v	29/20	150 MHz	110 MHz
trs_clipper.v	8/0	2.5 ns	4.8 ns
X9002.v	28/33	250 MHz	N/A
X7C9235.v	35/34	150 MHz	110 MHz
patho_trs.v	55/85	150 MHz	120 MHz

Notes:

The ser_scrambler2 and X9002 modules use features unique to the Virtex-II series.

Conclusion

Virtex-II and Spartan-II FPGAs can implement the SDI encoder function. Because the SDI encoder uses very few FPGA resources, it can be placed in the same device along with other related video functions resulting in a highly integrated design.

Three different implementations of the SDI scrambler are described in this application note. This will allow designers to trade-off clock rate versus FPGA area when creating an SDI encoder design. The half-bit rate scrambler is particularly well suited for use with the DDR support in the Virtex-II IOBs.

References

1. All the SMPTE standards referenced in this application note are available from The Society of Motion Picture and Television Engineers. These standards can be purchased at the SMPTE web site: http://www.smpte.org.
2. The ITU-R BT.601-5 standard can be purchased from the International Telecommunication Union at http://www.itu.int/itudoc/itu-r/rec/bt/.
3. The IEC 1179 standard is now called the IEC 61179 standard and can be purchased from the International Electrotechnical Commission at http://www.iec.ch/webstore.
4. Xilinx application note **XAPP288**: MicroBlaze and Multimedia Development Board: Serial Digital Interface (SDI) Video Decoder by John F. Snow.
5. Xilinx application note XAPP299: MicroBlaze and Multimedia Development Board: Serial Digital Interface (SDI) Ancillary Data and EDH Processors by John F. Snow.
6. Xilinx application note XAPP247: MicroBlaze and Multimedia Development Board: Serial Digital Interface (SDI) Physical Layer Implementation by John F. Snow.
7. Xilinx application note XAPP248: MicroBlaze and Multimedia Development Board: Digital Video Test Pattern Generators

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/02/01	1.0	Initial Xilinx release.