XAPP222 (v1.0) September 27, 2000

Designing Convolutional Interleavers with Virtex Devices

Author: Gianluca Gilardi and Catello Antonio De Rosa

Summary

The convolutional interleaver technique is used in telecommunication applications such as SDH and PDH radio systems, GSM and UMTS mobile communication systems, and point-to-multipoint radio systems to protect transmission channels from noise. On the transmit side, the convolutional interleaver parallelizes serial input data into *N*-bit words and shifts the data word through *N* delay lines. The delayed data is then shifted out through a PISO shift register for transmission. At the receiver, the incoming data stream is reconstructed with dual delay lines and shift registers.

The Virtex™ architecture is well suited to implement easy, fast, and efficient convolutional interleavers. With the SRL16 feature available in Virtex devices, very efficient shift registers can be implemented. The flexible shift registers vary in length from one to sixteen bits as determined by the address lines. In Figure 1, an SRL16 is used to implement a progressive delay line, thereby saving logic resources and producing the highest performance.

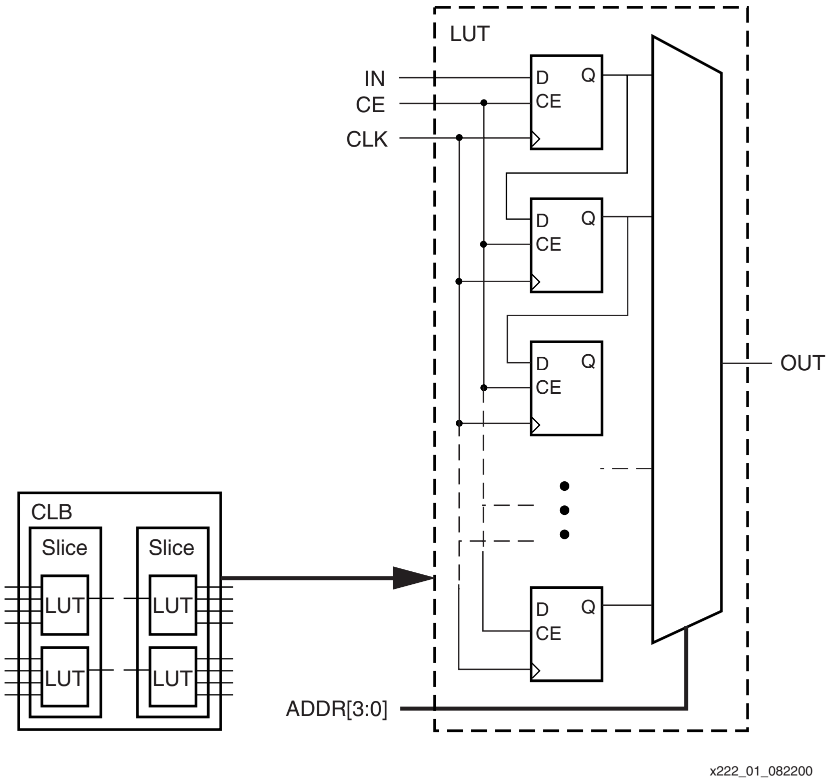


Figure 1: **SRL16**

Introduction

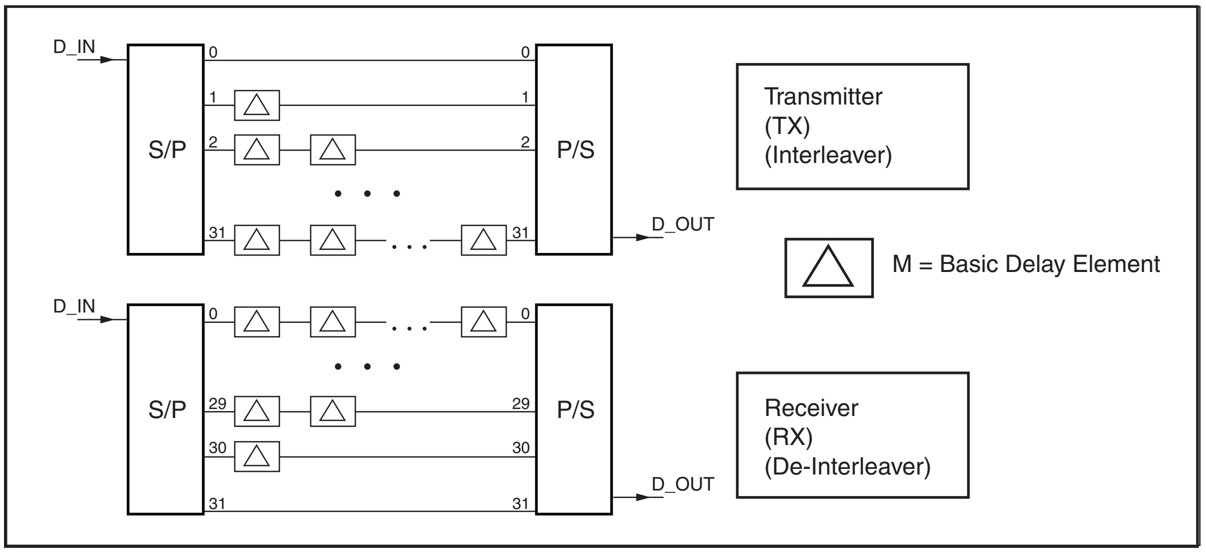
Data transmission systems must be protected from stochastic noise events on communication channels. Data scramblers (synchronous and asynchronous), check codes, error correction codes (Reed-Solomon, Viterbi, BCH), and interleavers (block or convolutional) are popular techniques for protecting data from noise. Interleavers are a popular choice for controlling impulse noise, which is characterized by high-power, short-duration bursts. Convolutional interleavers are often used in conjunction with Viterbi or Reed-Solomon codecs, since the load is dramatically reduced after the de-interleaver stage.

Convolutional Interleaver

The heart of a convolutional interleaver is a set of progressive delay lines. In Figure 2, the delay lines constitute two parts, one interleaver in the transmitter (TX) and one de-interleaver in the receiver (RX).

Consider a data stream, D_IN, feeding into the transmitter (TX). At a certain rate, this flow will be structured by a serial-in parallel-out (SIPO) register in a *B*-bit-wide bus (e.g., $B = 32$). The parallel flow then enters a progressive delay line where *M* is the delay unit. The first signal is not delayed, the second is delayed by *M*, the third is delayed by 2*M*, and so on until the *B*th signal is delayed by $(B-1)M$. All *B*-bit-wide signals are reserialized by a parallel-in, serial-out (PISO) register, and the serial output data flow, D_OUT, is ready for transmission.

On the receiving end, the receiver circuits are synchronized with the transmitter to reconstruct the data stream. In TX and RX circuits, a counter provides the correct timing and synchronization of events. Convolutional interleavers are conceptually similar to block interleavers, although they are more complex to implement. Convolutional interleavers are area efficient, using only half the density needed by block interleavers. In addition, the synchronization circuitry is simplified at the receiver, because the phases are *B* instead of *B* x *N*, where *B* is the number of block interleaver rows and *N* is the number of block interleaver columns.



X222_02_082200

Figure 2: **Convolutional Interleaver and De-interleaveer Schematic**

System Synchronization

In a radio system, a convolutional de-interleaver is generally put before a Viterbi decoder whose output provides the necessary synchronization. Consider a certain period of time, when the error rate is observed after the decoder processing. If in this period the error rate falls to zero, then synchronization is achieved. On the other hand, if this rate is not zero, a stop of one clock cycle to the RX synchronization counter must be provided. At most, after *B* steps, the synchronization phase is caught up.

System Architecture

In this example, the TX of the interleaver circuit has $B = 32$ phases, $M = 1$ delay units, and contains three elements: one 32-tap SIPO, one 32-bit progressive delay line (from 0 to 31 delay elements), and one 32-tap PISO to stream back serially formatted data for transmission. One 5-bit counter ($B = 32$) provides the correct delay adjustment, synchronizing the SIPO at the input of the delay line with acquisition by the PISO at the output of the delay line. An equation for the number (N_D) of the delay elements is then calculated:

$$N_D = B(B-1) / 2 = 496$$

The de-interleaver receiver is similar to the interleaver architecture with only the delay elements reversed (Figure 2). The receiver needs an external synchronization signal to enable the 5-bit counter ($B = 32$) to provide the correct circuit delay (similar to the transmitter). If this signal is not available, it is possible to use the output of the Viterbi decoder, as mentioned in the previous paragraph.

This circuitry is useful for a one-transmission data channel. With an 8-bit digital transmission, there are eight serial channels where an interleaver/de-interleaver protection system must be implemented.

Progressive Delay Line

Traditionally in FPGAs, the unit delay elements $M = 1$ (Figure 2) of the progressive delay line are implemented with flip-flops, making it a very register-intensive application. The additional resources needed to implement a real transmission system (for example, an 8-bit TX/RX system, 496 x 8 x 2 = 7936 flip-flops) traditionally favored ASIC solutions, but produced significant performance degradation.

The Virtex architecture redefines the suitability of FPGAs in transmission systems by implementing the delay elements in an exclusive SRL16 feature. The built-in programmable shift register enables one LUT to hold sixteen delay elements. In a system where $B = 32$, one LUT can implement from 1 to 16 delay lines. From 17 to 31 delay lines can be implemented with two LUTs per line, saving logic resources and maximizing overall performance. The only limiting factor is the physical LUT access time. In an 8-bit TX/RX system, for instance, the logic resources used are calculated:

$$(16 + (2 \times 15)) \times 8 \times 2 = 736 \text{ LUTs}$$

The Virtex solution uses almost ten times less logic resources then a traditional flip-flop solution.

SIPOs, PISOs, and Further Area Reduction

Generally, SIPOs and PISOs are implemented with flip-flops. Another technique, depending on logic resources and memory priorities, is to use block RAM.

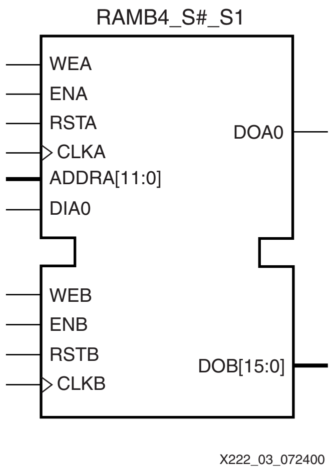


Figure 3: **Dual-Port Block RAM Memory**

The available block RAM primitives are shown in Figure 3 and Table 1. With the true dual-port capability of Virtex block RAM, A and B are used effectively as two independent ports, without any additional logic, thus allowing data flow width conversions. For instance, a 16-tap SIPO/PISO is a conversion 1:16 or 16:1.

Table 1: **Available Primitives**

Primitive	Port A Width	Port B Width
RAMB4_S1_S1	1	1
RAMB4_S1_S2	1	2
RAMB4_S1_S4	1	4
RAMB4_S1_S8	1	8
RAMB4_S1_S16	1	16

Comparison

Traditional (Flip-Flop) vs. Virtex Exclusive (SRL16) Solutions

Table 2 shows the efficiency of implementing this application in the Virtex architecture by comparing an eight channel interleaver/de-interleaver system. The interleaver and de-interleaver are implemented with B = 32 delay stages. This design was implemented for three Xilinx families: XC5210 (flip-flops), XC4062XL (delay elements by Core Gen), and XCV200 (SRL16s) devices.

Table 2: **Convolutional Interleaver Across Xilinx Families**

Device	CLB Array (R x C x LC)	LUTs	Flip-Flops	Used Area	SRL16
XC5210	18 x 18 x 4	514	8972	8 FPGAs	0
XC4062XL	48 x 48 x 2	1814	2040	1090 CLBs	0
XCV200	28 x 42 x 4	509	1036	1352 Slices	736

Additionally, Table 3 shows the final results of the same interleaver implemented on an XCV400E device, the first with flip-flops and the second with SRL16s.

Table 3: **SRL16 vs. Flip-Flop Implementation: Resource Usage in the Progressive Delay Line**

	CLB Array (R x C x LC)	LUTs	Flip-Flops	SRL16	Slice Usage	Max Frequency (-6)	Max Frequency (-8)
Flip-Flop	40 x 60 x 4	509	8972	0	4800 (100%)	102 MHz	131 MHz
SRL16	40 x 60 x 4	509	1036	736	1352 (28%)	128 MHz	160 MHz

The Virtex architecture allows the designer to implement a high-performance convolutional interleaver/de-interleaver with a small amount of FPGA resources by utilizing the SRL16 technique.

Synthesizable HDL Code Reference Design

This section of the application note describes a hierarchical synthesizable implementation convolutional interleaver/de-interleaver protection system for an eight-channel TX/RX radio system in Virtex devices with SRL16s. Complete VHDL code is available as a reference design (xapp222.zip and xapp222.tar.gz). The following design modules are included:

- **`TOP.vhd`**: Top file; example of instantiation of an 8-channel interleaver/de-interleaver system
- **`MULTI_CH.vhd`**: parametric VHDL for *N*-bit-wide interleaver/de-interleaver system
- **`TX_INTER`**: convolutional interleaver with *B* = 32 delay stages
- **`RX_INTER`**: convolutional de-interleaver with *B* = 32 delay stages

Foundation 3.1i ISE Convolutional Interleaver Test Circuit

This archived project can be loaded into the Xilinx Foundation 3.1i ISE front end tools. The circuit in Figure 5 shows an example of how to control a convolutional interleaver under ideal conditions. The delay element *M* is equal to the system clock in this case, giving a relative delay value of 1. This simplifies the synchronization circuitry for this example; in a real communication system, the synchronization is handled by the Viterbi decoder or other control mechanism. Using this example, the designer has the ability to use the convolutional interleaver on specific designs.

- **`MASTER.sch`**: top-level schematic showing the connectivity between all the components. In simulation the user needs to supply a serial data vector and an active high GO signal
- **`C5CE.sch`**: 5-bit counter
- **`CONTROLBOX.sch`**: a simple state machine to control the synchronization of the convolutional interleaver. The FSM implements the state transition diagram (Figure 4).

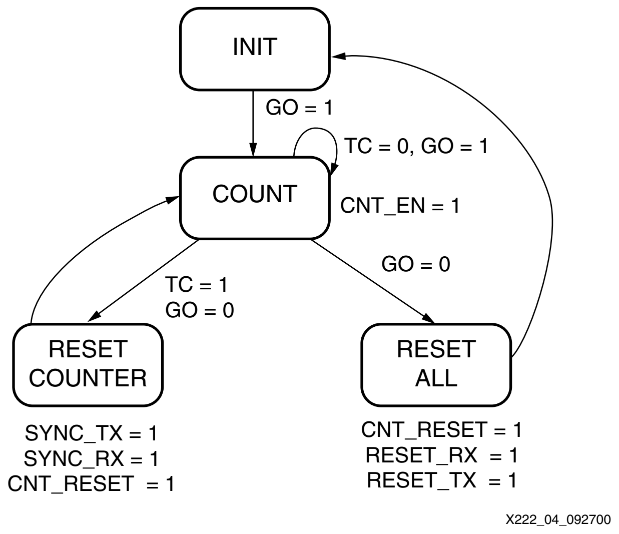


Figure 4: **State Machine**

- **`SIPO.v`**: serial-in parallel-out shift register. Provided for the designer to see the output of the receiver as a parallel word instead of as a serial data stream
- **`RX_INTER.vhd`**: VHDL module describing the 32-delay tap convolutional de-interleaver
- **`TX_INTER.vhd`**: VHDL module describing the 32-delay tap convolutional interleaver

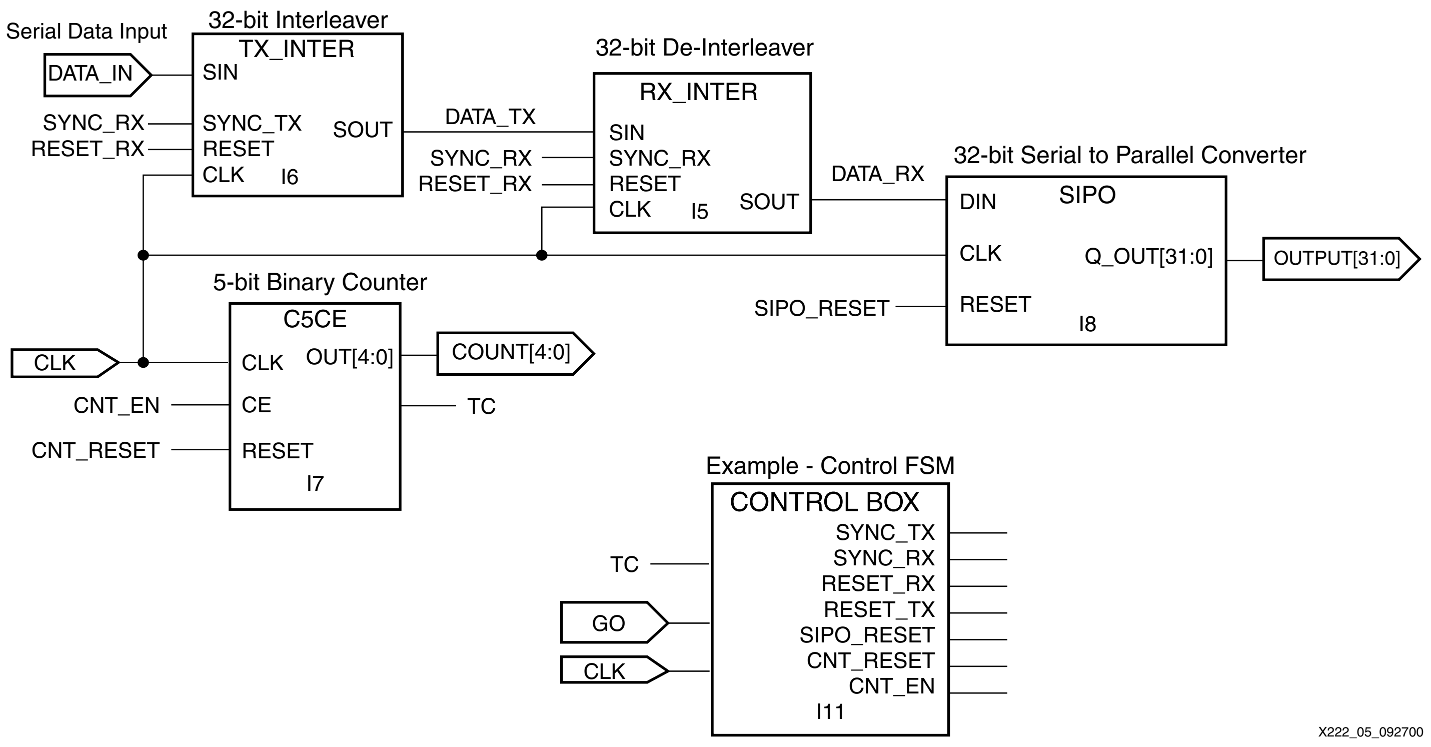


Figure 5: **Test Circuit**

Conclusion

This application note and reference design clearly demonstrate the efficiency of the SRL16 technique. This exclusive Virtex architectural feature is the key advantage when designing a convolutional interleaver/de-interleaver for TX/RX radio systems.

Using the SRL16 feature instead of flip-flops to make variable-delay elements saves resources and maximizes performance. Additionally, the true dual-port capability of Virtex block RAM allows further area reduction by making SIPOs and PISOs using the capability to set different widths for port A and port B. This easy, fast, and efficient solution is feasible only with the Virtex architecture. Other FPGAs utilize traditional logic-consuming, register-intensive approaches.

References

1. Xilinx Inc., DS022, ***Virtex-E 1.8V Field Programmable Gate Arrays Data Sheet***
2. Xilinx Inc., DS003, ***Virtex-E 2.5V Field Programmable Gate Arrays Data Sheet***
3. Xilinx Inc., DS001, ***Spartan®-II 2.5V Field Programmable Gate Arrays Data Sheet***
4. Xilinx Inc., XAPP210, ***Linear Feedback Shift Registers in Virtex Devices***
5. Xilinx Inc., XAPP130, ***Using the Virtex Block SelectRAM™+ Features***

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/27/00	1.0	Initial Release