XILINX®

XAPP217 (v1.0) June 29, 2000

Gold Code Generators in Virtex Devices

Author: Maria George, Mujtaba Hamid, and Andy Miller

Summary

Gold code generators are used extensively in Code Division Multiple Access (CDMA) systems to generate code sequences with good correlation properties. This application note describes the implementation of Gold code generators in Virtex™ and Spartan®-II devices. The Gold code generators use efficiently implemented Linear Feedback Shift Registers (LFSRs) in Virtex and Spartan-II devices using the SRL16 macro.

Introduction

In a multi-user CDMA system several forms of "Spread Spectrum" modulation techniques are used. The most popular is the Direct Sequence Spread Spectrum (DS-SS). In this form of modulation each user signal is uniquely coded and spread across a wide band of transmission frequencies. Pseudo-random Noise (PN) sequences that are orthogonal to each other are used to code the user signals. Two sequences are considered orthogonal when their cross-correlation coefficient is zero. These PN sequences are generated using Gold code generators. The basic functional blocks for Gold code generators are LFSRs.

The SRL16 (Shift Register Look-Up-Table) macro in Virtex and Spartan-II devices are used to implement LFSRs thereby reducing FPGA resource utilization. The length of the shift register can be set to any value from one to 16. The shift register can be set to either a fixed/static length or dynamically adjusted by controlling the four address inputs A[3:0].

PN Sequences in DS-SS Systems

A Pseudo-random Noise (PN) sequence/code is an orthogonal, finite length, binary sequence. Ideally, a PN sequence should be orthogonal to every time shifted version of itself.

There are three uses for PN sequences in DS-SS applications:

1. Spreading the bandwidth of the modulated signal over a wide radio spectrum.
2. Uniquely coding the different user signals that occupy the same transmission bandwidth in a multi-access system.
3. Synchronization for W-CDMA systems where there is no global timing reference.

In order to achieve these objectives, the coding sequences require special correlation properties referred to as auto correlation, and cross correlation.

Auto Correlation

Auto correlation is a measure of how well a signal f (t) can differentiate between itself and every time-shifted variant of itself. Auto correclation for a finite, discrete signal is defined in Equation 1 where T is a time delay and L is the sequence length.

$$r_{xx}(T) = \sum_{n=0}^{L} x(n)x(n-T) \quad \text{(unnormalized)} \qquad (1)$$

This equation provides three distinct pieces of information. A positive correlation is indicated when a signal is difficult or impossible to distinguish from a time-shifted version of the original signal. A negative correlation is indicated when a signal can be distinguished from the original signal. A correlation of zero indicates a signal that is orthogonal to a time-shifted version of itself.

Consider a data word of period seven bits at time δt = 0 (Table 1). If the 7-bit code were to be repeating within a discrete system then there are only six time-shifted replicas of the word, (shown in Table 1 for time δt = 1 to δt = 6). If each bit of the original (δt = 0) is compared with each bit of every time-shifted replica, then there are a number of agreements (A), and disagreements (D), that when subtracted provide a measure of how closely the two words match (correlate).

In Table 1, the sequence "1110010" has a good auto correlation property as it provides a clear difference in the correlation value between itself and any time-shifted variant of itself.

In Table 2, the sequence "1111000" has the same number of bits, but the auto correlation property is not as good as there are some clear rejections of a match (correlation value = –5), and there are some "fuzzy" conditions where the time-shifted replica almost matches, (correlation value = 3).

Table 1: **Auto Correlation Example**

Sequence	Time Shift	(A)	(D)	(A – D)
1110010	δt = 0	7	0	7
0111001	δt = 1	3	4	–1
1011100	δt = 2	3	4	–1
0101110	δt = 3	3	4	–1
0010111	δt = 4	3	4	–1
1001011	δt = 5	3	4	–1
1100101	δt = 6	3	4	–1

Table 2: **Auto Correlation Example**

Sequence	Time Shift	(A)	(D)	(A – D)
1111000	δt = 0	7	0	7
0111100	δt = 1	5	2	3
0011110	δt = 2	3	4	–1
0001111	δt = 3	1	6	–5
1000111	δt = 4	1	6	–5
1100011	δt = 5	3	4	–1
1110001	δt = 6	5	2	3

Cross Correlation

Cross correlation is defined as the correlation between two different signals. Cross correlation is also calculated by subtracting the disagreements from the agreements, between two different sequences as opposed to the time-shifted replicas of the same signal. Cross correlation for finite length descrete signals is defined in Equation 2.

$$r_{xy}(T) = \sum_{n=0}^{L} x(n)y(n-T) \quad \text{(unnormalized)} \qquad (2)$$

Non-Interfering Codes for User Signals

In a CDMA system, each one of the multiple user signals in the receiver is assigned a unique PN code that behaves like a "key". From the examples in Table 1 and Table 2 it is evident that some sequences of the same length have better cross-correlation properties than others and these special PN sequences are the ones used to code user signals in the system. Ideally, all the PN sequences used for coding are orthogonal to each other. Preferred sequences are a small subset of the possible approximate (small cross-correlation) m-sequences, Maximal Length Sequences (L). Combining two preferred sequences together through an XOR gate generates a Gold code.

It is important to use a set of PN sequences that have a small cross correlation between each other in order to reduce an effect called co-channel interference. If the cross correlation between two PN sequences or "keys" is not small, there is a possibility that data coded from one user is incorrectly identified and assigned to another user because the two keys had a reasonable correlation.

Research on small cross correlation PN sequences, have been carried out by many individuals but code sets identified by Kasami, R. Gold and Walsh are used throughout the IS-95 and UMTS W-CDMA systems.

When reviewing technical system specifications that require the use of PN generators or LFSRs, it is usual to find comments such as "codes from the Kasami set of Length…" or "Gold sequences generated with two polynomials of degree 41", or simply "$x(n) = 1 + X^3 + X^7$".

The next section reviews some of the terminology associated with LFSRs in order to help match the system level definition to an architectural implementation.

LFSR Terminology

The basic functional block in Gold code generators are LFSRs. LFSRs sequence through $(2^N - 1)$ states, where N is the number of registers in the LFSR. The contents of the registers are shifted right by one position at each clock cycle. The feedback from predefined registers or taps to the left most register are XOR-ed together.

LFSRs have several variables:

- The number of stages in the shift register.
- The number of taps in the feedback path.
- The position of each tap in the shift register stage.
- The initial starting condition of the shift register often referred to as the "FILL" state. In the case of LFSRs with an XOR feedback, the "FILL" value must be a non-zero value to avoid the LFSR locking up in the next state.

Shift Register Length (N)

This is often referred to as the degree, and in general, the longer the shift register, the longer the duration of the PN sequence before it repeats. For a shift register of fixed length N, the number, and duration of the sequences that it can generate, are determined by the number, and position of taps used to generate the "parity" feedback bit.

LFSR Implementation

There are two implementation styles of LFSRs, Galois implementation and Fibonacci implementation.

Galois Implementation

As shown in Figure 1, the data flow is from left to right and the feedback path is from right to left. The polynomial increments from left to right with X^0 term ("1" in the polynomial) as the first term in the polynomial. This polynomial is referred to as a Tap polynomial and indicates which taps are to be fed back from the shift register. The XOR gate is in the shift register path, therefore, the Galois implementation is also known as an in-line, modular type, or M-type LFSR.

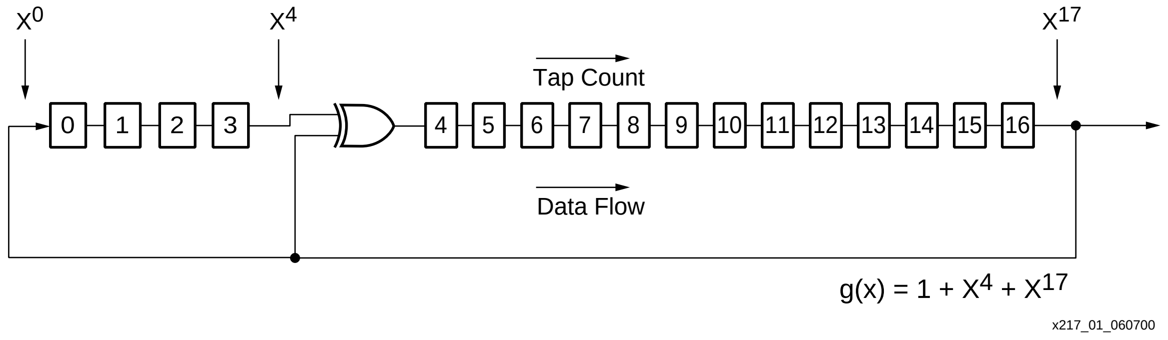


Figure 1: **Galois Implementation**

Fibonacci Implementation

In Figure 2 the data flow is from left to right and the feedback path is from right to left, similar to the Galois implementation. However, the Fibonacci implementation polynomial decrements from left to right with X^0 as the last term in the polynomial. This polynomial is referred to as a Reciprocal Tap polynomial and the feedback taps are incrementally annotated from right to left along the shift register. The XOR gate is in the feedback path, therefore, the Fibonacci implementation is also known as an the Out-of-Line, Simple-Type or (S-Type) LFSR.

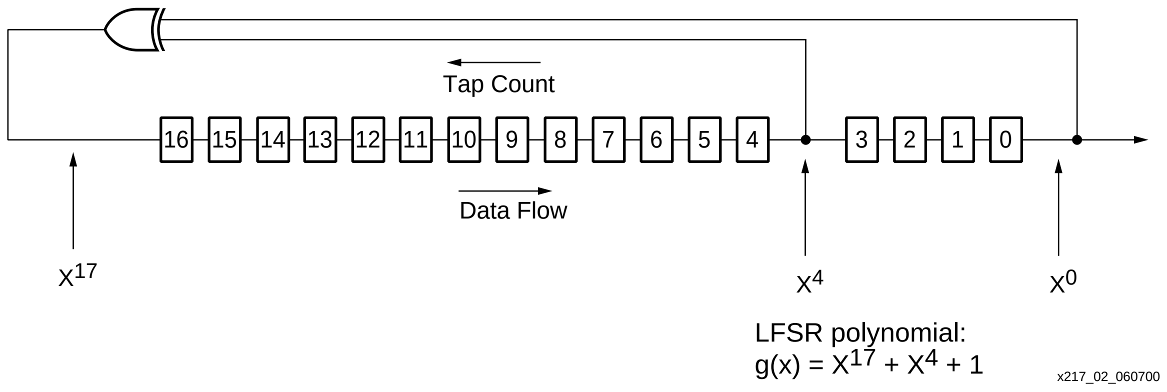


Figure 2: **Fibonacci Implementation**

Shift Register Taps

The combination of taps and their location is often referred to as a polynomial, and expressed as:

$$P(x) = X^7 + X^3 + 1$$

Various conventions are used to map the polynomial terms to register stages in the shift register implementation. The convention used in this application note is consistent with the convention used in the CDMA UMTS specification.

In the polynomial $P(x) = X^7 + X^3 + 1$, the trailing "1" represents X^0, which is the output of the last stage of the shift register. X^3 is the output of register stage 3 and X^7 the output of the XOR.

A couple of points to note about LFSRs and the polynomial used to describe them are:

- The last tap of the shift register is the leading "1" and always used in the shift register feedback path.
- The length of the shift register can be deduced from the exponent of the highest order term in the polynomial.
- The highest order term of the polynomial is the signal connecting the final "XOR" output to the shift register input. It does not feed back into the parity calculation along with the other taps identified in the polynomial.

Maximal Length Sequences (L)

A maximal length sequence for a shift register of length N is referred to as an m-sequence, and is defined as:

$$L = 2^N - 1$$

e.g., An eight stage LFSR will have a set of m-sequences of length 255.

Correlation Properties

There are many combinations of taps that produce small cross correlation m-sequences. Consequently it is possible to define a set of taps that produce a collection of small cross correlation m-sequences for a constant length shift register. Kasami, Walsh, and Gold are recognized for the identification of small cross correlation maximal length PN codes.

The number of independent m-sequences (S), for a given length of shift register is defined by:

$$S \le (L - 1) \div N$$

Gold Code Generators

Gold code generators were initially presented in 1967 by R. Gold. *He suggested that sets of small correlation PN codes could be created by Modulo 2 addition of the results of two LFSRs, primed with factor codes.* The result is a set of codes with correlation properties ideally suited to distinguish one code from another in a spectrum full of coded signals. These codes are known as Preferred Pair Gold Codes. They are generated by XORing the outputs of two same-length LFSRs primed with specific Fill values from two factor codes.

Figure 3 shows an implementation of a Gold code generator. Two same-length LFSRs loaded with paired factor codes are XOR'd to create a new family of codes suited for use in CDMA systems. At the system level, a Gold code generator is usually described by two polynomials indicating the LFSR structure to be implemented.

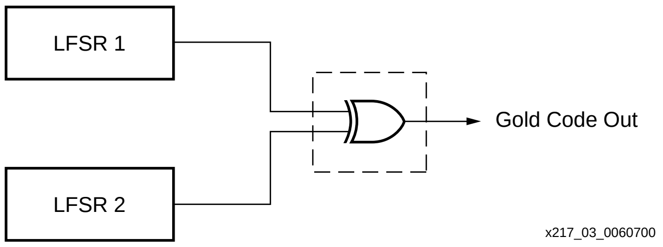
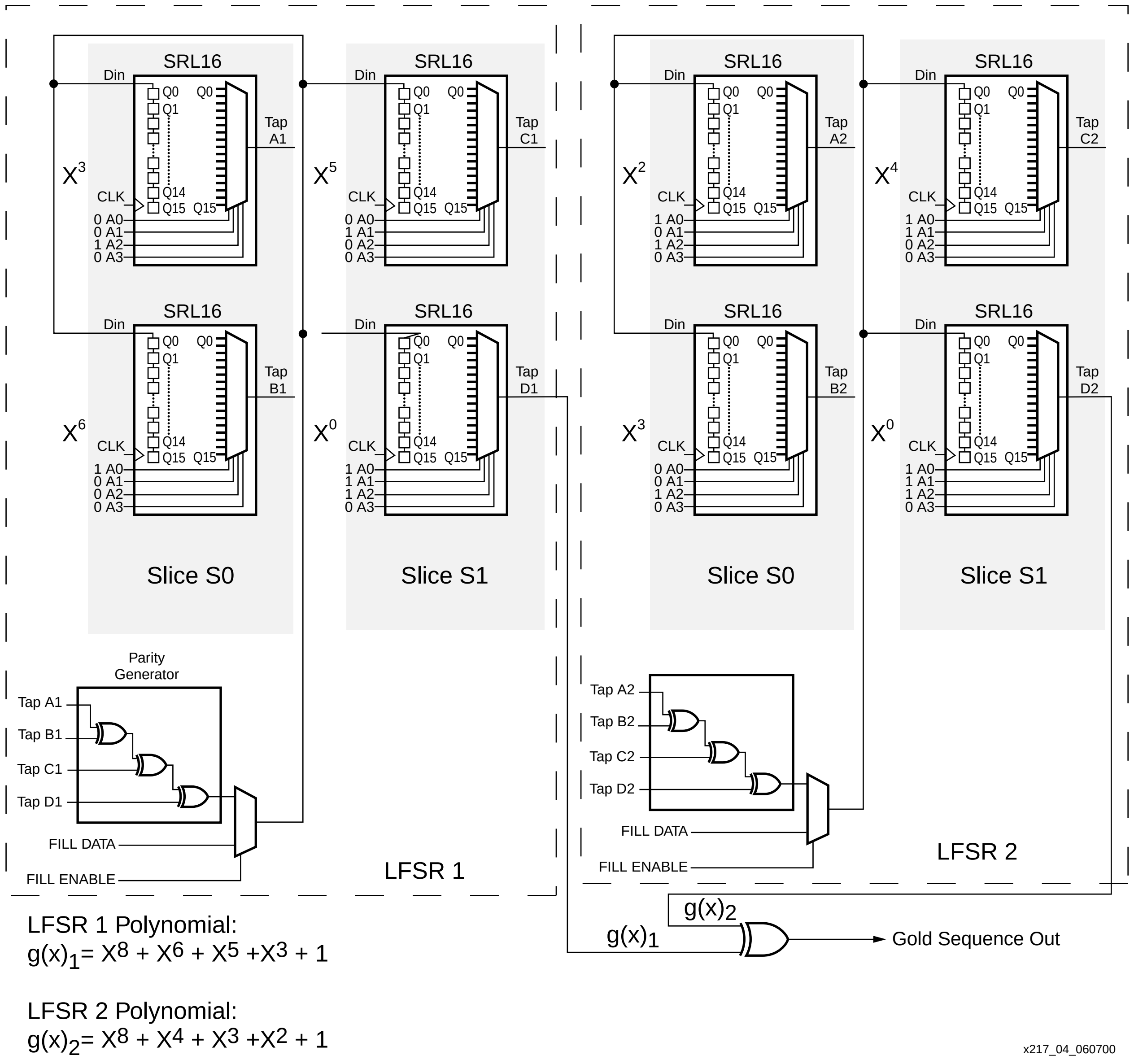


Figure 3: **Gold Code Generator**

Gold Code Generators using the LFSRs Implemented in Virtex Devices

A 16-bit LFSR uses one slice in a Virtex device. A Virtex slice is comprised of one 4-input look-up table (LUT), carry logic and a storage element. A Virtex configurable logic block (CLB) contains two slices. The area occupied by Gold code generators in Virtex devices are a function of the number of stages and taps used by the LFSR.

Figure 4 is an 8-stage 4-tap Gold code generator implemented in 6.5 slices (3.25 Virtex CLBs). Figure 5 demonstrates a 41-stage, 2-tap Gold code generator implemented in 5.5 slices (2.75 Virtex CLBs).



LFSR 1 Polynomial:
$g(x)_1 = X^8 + X^6 + X^5 + X^3 + 1$

LFSR 2 Polynomial:
$g(x)_2 = X^8 + X^4 + X^3 + X^2 + 1$

Figure 4: **8-Stage, 4-Tap Gold Code Generator**

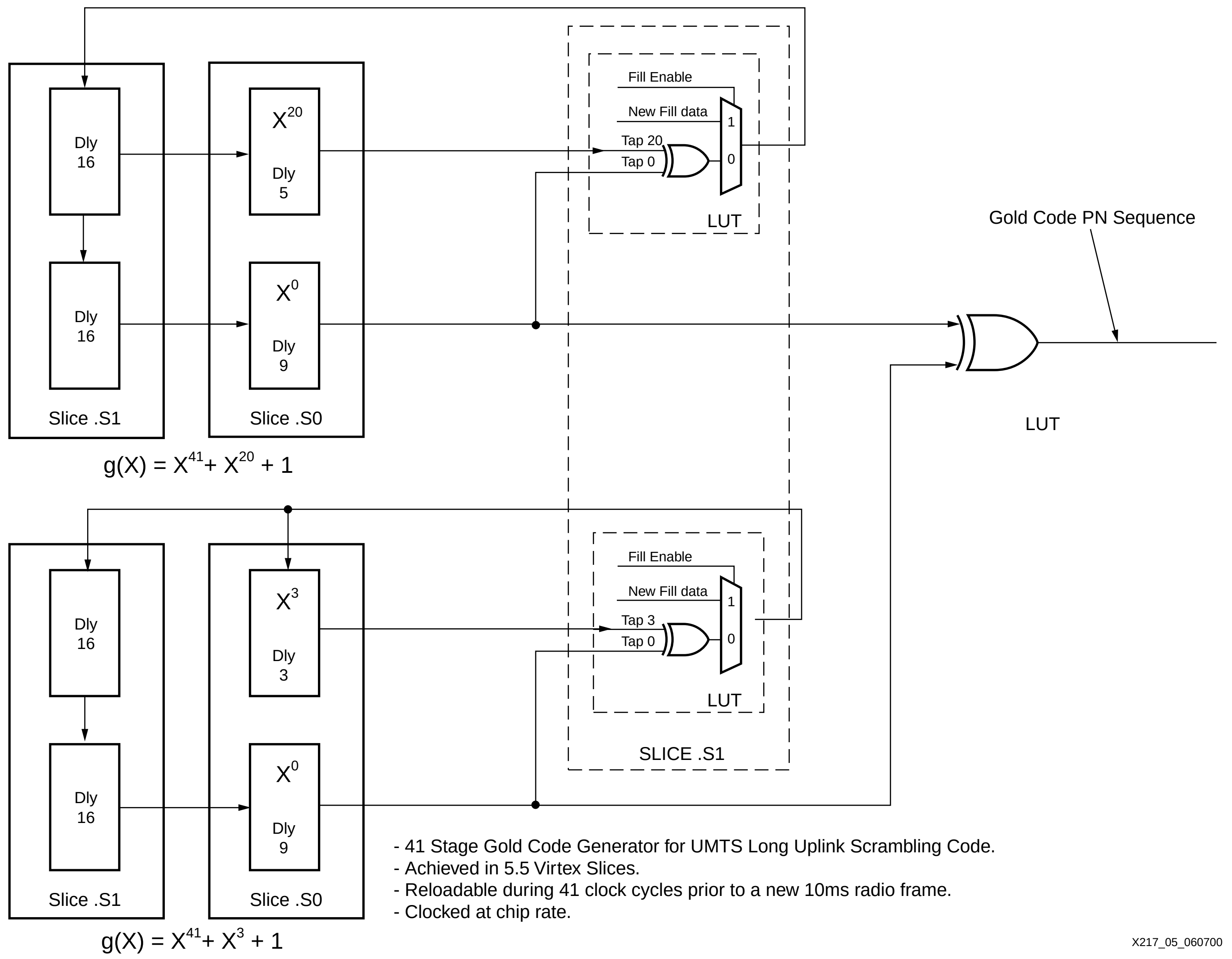


Figure 5: **41-Stage, 2-Tap Gold Code Generator**

Gold Code Generator HDL Code

The Gold code generator reference design was written in both VHDL and Verilog HDL. The files are available on the Xilinx web site at xapp217.zip or xapp217.tar.gz.

The code replicates the logic shown in Figure 5. The output of the Gold code generator is obtained by XORing Tap 41 of the two LFSRs. The design was targeted for an XCV50-6BG256 device. The performance results are listed in Table 3. The code was tested to work with current versions of Express, Exemplar, and Synplify. For both VHDL and Verilog code, the design is in two hierarchical levels. This makes the code readable and produces more efficient debugging and verification.

In the reference design, Virtex SRL16E components were inferred to achieve the most efficient implementation results. The SRL16 is a shift register LUT with four inputs to select the length of the output signal. The SRL16 component can be instantiated in code but doing so limits the ability to quickly modify the functionality.

To ensure that the SRL component is inferred, follow the required syntax. Synthesis tools recognize this syntax and infer the SRL16 component for Virtex FPGAs. If, for any reason, the code is written differently the output netlist (written by the Synthesis tool) should check and verify the presence of the the SRL16 components. An example of the syntax to correctly infer the SRL 16 component is available on the Xilinx website at **http://www.xilinx.com/techdocs/7822.htm**. For readablility, each LFSR implementation is in a separate block. If there are any problems after following the syntax in the above listed solution, contact Xilinx Technical Support at http://support.xilinx.com.

The code was simulated on MTI's Modelsim simulator using the TCL interface, therefore, no testbench was used. On a simulator supporting stimulus using HDL code only, create the testbench (HDL) file to verify functionality .

Table 3: **Utilization/Performance**

	Design Implementation	Synopsys FPGA Express v3.3	Synplicity Synplify v5.31	Exemplar Leonardo v1999.1h
Utilization	SRL16	5 slices	5 slices	5 slices
Performance	SRL16	123 MHz	128 MHz	176 MHz

The example in this document is for creating a 41-stage Gold code generator. Each LFSR is a 41-stage, 2-Tap LFSR implemented using SRL16Es.

Conclusion

Implementing Gold code generators in Virtex devices using the SRL macro is efficient in terms of FPGA utilization. For example, a 41-stage Gold code generator can be realized in just six Virtex slices.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/29/00	1.0	Initial Xilinx release.