An Attribute-Programmable PRBS Generator and Checker

Author: Daniele Riccardi and Paolo Novellini

XAPP884 (v1.0) January 10, 2011

Summary

In serial interconnect technology, it is very common to use pseudorandom binary sequence (PRBS) patterns to test the robustness of links.[1] Behind this choice is the fact that every PRBS pattern has a white spectrum in the frequency domain or, alternatively, a delta-shaped auto-correlation function in the time domain. This application note describes a PRBS generator/checker circuit where the generator polynomial, the parallelism level, and the functionality (generator or checker) are programmable via attributes.

The sections Circuit Description and Pinout, Standard Polynomials, page 2, and Design Example, page 3 are intended for users who want to know how to use the PRBS generator and checker. The final section, PRBS Sequences: Theoretical Background, page 5, is a theoretical background on PRBS sequences, their generators, and how to select them to obtain optimal spectral properties.

Circuit Description and Pinout

Table 1 shows the attributes available to configure the PRBS module, while Table 2 specifies the port list. The circuit that generates or checks a PRBS sequence is based on a linear feedback shift register (LFSR). In this application note, the bit sequence is indicated by the term PRBS, and the circuit that generates or checks it is indicated by the term LFSR.

Table 1: **PRBS Generator/Checker Attributes**

Attribute	Type	Description
CHK_MODE	Boolean	**FALSE**: The module generates a PRBS sequence. **TRUE**: The module checks a PRBS sequence.
INV_PATTERN	Boolean	**FALSE**: The generated pattern is not inverted or the checker does not invert the pattern before checking it. **TRUE**: The generated pattern is inverted or the checker inverts the pattern before checking it.
POLY_LENGTH	Natural (2 to 63)	This is the trinomial degree.
POLY_TAP	Natural (1 to 62)	This attribute specifies the trinomial for which the backward counter in the Galois Field is generated. See How to Choose the Generator Polynomial, page 5 and How to Construct LFSRs from a Given Polynomial, page 5.
NBITS	Natural (1 to 512)	This is the parallelism level for the generator and checker.

---

1. International Telecommunication Union Telecommunication Standardization Sector (ITU-T) Recommendation O.150 is a good example of this. All ITU-T test patterns are PRBS sequences.

Table 2: **PRBS Generator/Checker Ports**

Port	Direction	Description
RST	IN	Reset
CLK	IN	Clock
DATA_IN	IN from NBITS (–1 to 0)	In check mode, this is the deserialized data to be checked. The width of DATA_IN is programmed by the attribute OUT_BITS. Bit 0 is the oldest. In generate mode, this input is used to insert an error in the corresponding output bit.
EN	IN	Enable for all internal synchronous processes.
DATA_OUT	OUT from NBITS (–1 to 0)	In generate mode, this is the generated PRBS. In check mode, any bit set to 1 indicates a bit error on the corresponding line.

The PRBS module can be configured to be a checker or a generator via the CHK_MODE attribute. In generate mode, the DATA_IN port is used to insert an error on the corresponding bit. This means that every bit can be individually inverted before being sent on DATA_OUT. In check mode, the DATA_OUT port reveals whether or not an error has occurred. When an error is detected, the corresponding line is set to 1. The OUT_BITS attribute specifies the parallelism level for both the checker and the generator functionality.

The PRBS module used as a self-synchronizing checker raises roughly three lines for each error inserted on the line. This is a fully synchronous circuit where all processes are controlled by CLK. EN is the enable signal for all internal processes. The convention for bit ordering is:

- In generate mode, the least significant bit is serialized first.
- In check mode, the least significant bit is the oldest received.

This is the same convention used in all Xilinx® transceivers starting with the Virtex®-4 family.

Standard Polynomials

This section summarizes the most often used PRBS sequences from ITU-T Recommendation O.150. Table 3 contains, for each listed polynomial, the PRBS module attribute configuration to generate or check the polynomial. Inversion can be easily set using the INV_PATTERN attribute.

Table 3: **Configuration for PRBS Polynomials Most Used to Test Serial Lines**

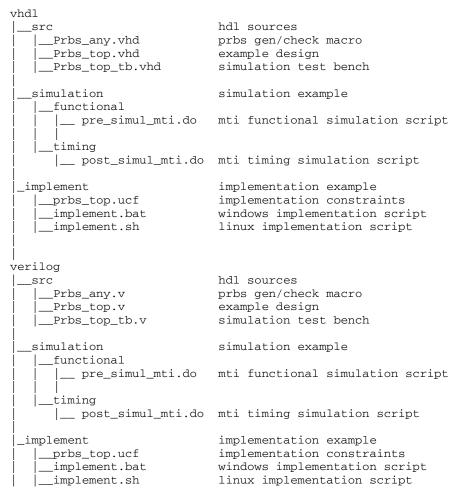| Polynomial	Reference Standard	POLY_LENGTH	POLY_TAP	INV_PATTERN
$x^7 + x^6 + 1$	Not standard[1]	7	6	TRUE
$x^9 + x^5 + 1$	ITU-T O.150	9	5	FALSE
$x^{11} + x^9 + 1$	ITU-T O.150	11	9	FALSE
$x^{15} + x^{14} + 1$	ITU-T O.150	15	14	TRUE
$x^{17} + x^{14} + 1$	OIF-CEI-P-02.0	17	14	FALSE
$x^{20} + x^3 + 1$	ITU-T O.150	20	3	FALSE
$x^{23} + x^{18} + 1$	ITU-T O.150	23	18	TRUE
$x^{29} + x^{27} + 1$	ITU-T O.150	29	27	TRUE
$x^{31} + x^{28} + 1$	ITU-T O.150 / OIF-CEI-02.0	31	28	TRUE

Notes:

1. The PRBS7 polynomial is not a telecommunications standard, but it is typically implemented in test equipment because it is very similar to 8B/10B-encoded patterns.

The LFSR implemented by the module is a backward counter in the Galois field defined by the corresponding polynomial in the first column. This is a conventional choice; by using this approach, the POLY_LENGTH and POLY_TAP attributes are equal to the maximum and medium degrees in the generator polynomial. The lowest degree is always 1.

Design Example

The downloadable archive file `xapp884.zip` includes the design example and the scripts for simulation and implementation. The directory structure is:

```
vhdl
|__src                     hdl sources
|  |__Prbs_any.vhd         prbs gen/check macro
|  |__Prbs_top.vhd         example design
|  |__Prbs_top_tb.vhd      simulation test bench
|
|__simulation              simulation example
|  |__functional
|  |  |__ pre_simul_mti.do  mti functional simulation script
|  |  |
|  |__timing
|     |__ post_simul_mti.do  mti timing simulation script
|
|_implement                implementation example
|  |__prbs_top.ucf         implementation constraints
|  |__implement.bat        windows implementation script
|  |__implement.sh         linux implementation script
|
|
verilog
|__src                     hdl sources
|  |__Prbs_any.v           prbs gen/check macro
|  |__Prbs_top.v           example design
|  |__Prbs_top_tb.v        simulation test bench
|
|__simulation              simulation example
|  |__functional
|  |  |__ pre_simul_mti.do  mti functional simulation script
|  |  |
|  |__timing
|     |__ post_simul_mti.do  mti timing simulation script
|
|_implement                implementation example
|  |__prbs_top.ucf         implementation constraints
|  |__implement.bat        windows implementation script
|  |__implement.sh         linux implementation script
```

Running the Simulation

In this simulation example, a PRBS generator implements a backward counter in GF(2^{23}) with parallelism 1. The implemented polynomial is the standard $x^{23} + x^{18} + 1$ from ITU-T O.150. The data stream is sent to two different deserializers, x8 and x9. The deserialized data is then applied to two checkers. The block diagram of the overall simulation is shown in Figure 1.

The example design includes scripts to run the functional or timing simulation. For more details, refer to the readme file supplied with the reference design. The testbench used for the design is shown in Figure 1.
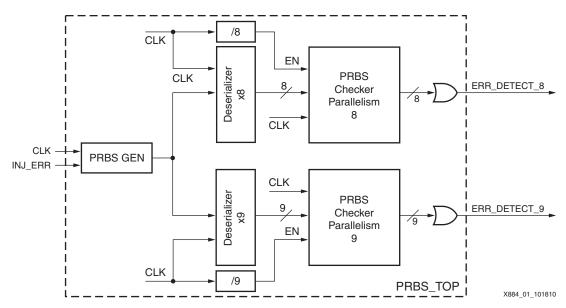
Figure 1: **Block Diagram of Testbench for PRBS Generator/Checker**

The results of the simulation are shown in Figure 2. Both checkers confirm that no errors are detected until INJ_ERR is pulsed.
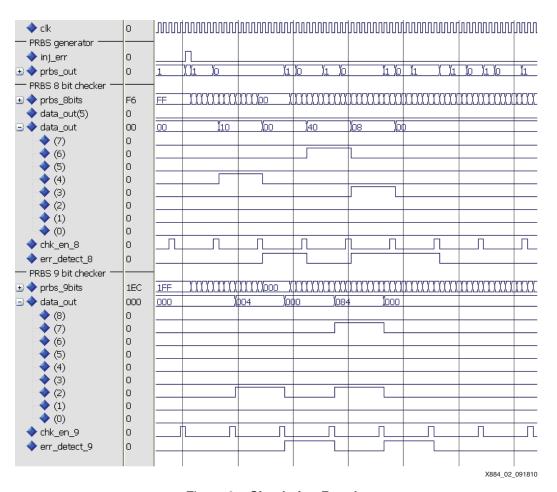

Figure 2: **Simulation Results**

At the beginning of the simulation, an error is intentionally injected in the line by pulsing INJ_ERR. This error is immediately detected, and both err_detect_8 and err_detect_9 are pulsed by the two independent PRBS checkers. The three blocks instantiated in the simulation testbench in Figure 1 (one generator and two checkers) are always the same component, but with a different attribute set.

Running the Implementation

To implement the design example, the appropriate implementation script should be run. To change the target family or size, the part name in the script should be modified before being run. Any Virtex or Spartan® family can be selected.

PRBS Sequences: Theoretical Background

This section discusses basic finite field theory, explaining how a polynomial is related to its corresponding LFSR implementations and how to select a PRBS of non-standard length with optimal spectral properties. For a PRBS to have good spectral properties, it must have an optimally flat power spectrum in the frequency domain. The user can verify that all these considerations were applied when defining the standard polynomials described in Standard Polynomials, page 2.

How to Choose the Generator Polynomial

In most cases, a PRBS module is configured with a standard polynomial. To configure the module to generate or check a standard polynomial, see Standard Polynomials, page 2.

When a standard polynomial is not well suited to the user's application requirements, or when a polynomial has been defined as a standard subsequent to the publication of this application note, this section can be consulted to learn how to choose a different polynomial.

The term "PRBS" is the sequence that is generated; the term "LFSR" is the circuit that generates or checks the PRBS sequence. Every time a polynomial of degree m is selected as a generator, the maximum period length obtainable in terms of PRBS sequence is $2^m - 1$.

However, not all polynomials of degree m guarantee that the maximum period length can be reached. Therefore, the designer typically looks for criteria to select a generator that guarantees a maximum-length PRBS output. In finite field theory, this is guaranteed by selecting a polynomial that is primitive.

Note: For a definition of "primitive polynomial" and an introduction to finite field theory, refer to *Error Control Coding* [Ref 1] or *Error-Correcting Codes* [Ref 2].

For every degree m, there are many primitive polynomials, especially as m becomes high. From among the many primitive polynomials of the same degree, the user should select one polynomial that involves the minimum number of feedback connections in order to simplify the hardware as much as possible. These polynomials are called trinomials, because they are of the form:

$$x^n + x^d + 1 \qquad\qquad \textit{Equation 1}$$

Where $d < n$.

A list of primitive trinomials can be found in *Error-Correcting Codes* [Ref 2].

How to Construct LFSRs from a Given Polynomial

A polynomial by itself does not define a specific hardware structure. Any primitive polynomial of degree m with coefficients in GF(2) defines an extension field of 2^m elements. An LFSR implements a counter in the extension field GF(2^m), which can count forward or backward, originating two PRBS sequences from the same polynomial. These sequences have the same spectral properties but are not compatible with each other.

The diagrams shown in Figure 3 and Figure 4 can be used to mechanically construct a forward counter and a backward counter, respectively, in any given polynomial. The signal flows in different directions and the flip-flops have a different orientation.
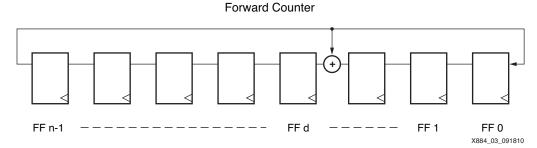
Forward Counter

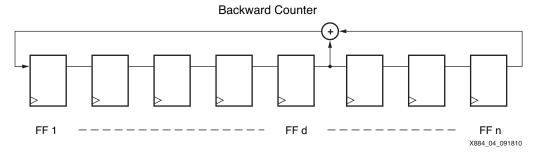*Figure 3:* **Forward Counter Implementation of $x^n + x^d + 1$**

Backward Counter

Figure 4: **Backward Counter Implementation of $x^n + x^d + 1$**

The convention used in this application note is to associate the polynomial to the backward counter in GF(2^m). If required to generate the forward counter for a given polynomial g(x), the reciprocal g'(x) should be constructed as a first step, according to Equation 2.

$$g'(x) = x^n g(x^{-1})$$

Equation 2

This can be proved to be primitive if g(x) is primitive. The backward counter LFSR for g'(x) is the same as the forward counter LFSR for g(x). Thus, with this simple rule, the backward and forward counter LFSRs can be implemented for any given primitive trinomial.

Example

In this example, a PRBS23 is to be implemented with parallelism of level 2 for the transmitter and level 8 for the checker. First, a primitive trinomial of degree 23 must be found. By looking at the tables in *Error-Correcting Codes* [Ref 2], it can be seen that Equation 3 is a good choice.

$$g(x) = x^{23} + x^{18} + 1$$

Equation 3

The backward counter LFSR can now be implemented by setting the feedback tap at 18. Setting the widths for the transmitter and receiver is immediate. The attribute settings for the transmitter and receiver are shown in Table 4.

Table 4: **Attribute Sets for PRBS Pattern Generator and Checker**

Attribute	Value	
	Generator Attribute Set	**Checker Attribute Set**
CHK_MODE	FALSE	TRUE
INV_PATTERN	FALSE	FALSE
POLY_LENGTH	23	23

Table 4: **Attribute Sets for PRBS Pattern Generator and Checker** *(Cont'd)*

Attribute	Value	
	Generator Attribute Set	**Checker Attribute Set**
POLY_TAP	18	18
NBITS	2	8

This is the choice in the ITU-T recommendation O.150 for the standard PRBS of degree 23. To implement the forward counter, the exercise is repeated by implementing the backward counter of Equation 4, which is the reciprocal of $g(x)$.

$$g'(x) = x^n g(x^{-1}) = x^{23} + x^5 + 1$$

Equation 4

Reference Design

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

https://secure.xilinx.com/webreg/clickthrough.do?cid=155046

The reference design checklist is shown in Table 5.

Table 5: **Reference Design Checklist**

Parameter	Description
General	
Developer Name	Daniele Riccardi and Paolo Novellini
Target Devices (Stepping Level, ES, Production, Speed Grades)	Xilinx FPGAs
Source Code Provided	Yes
Source Code Format	VHDL, Verilog
Design Uses Code/IP from Existing Application Note, Reference Designs, Third Party, or CORE Generator™ Software	No
Simulation	
Functional Simulation Performed	Yes
Timing Simulation Performed	Yes
Testbench Used for Functional and Timing Simulations	Yes
Testbench Format	VHDL, Verilog
Simulator Software/Version Used	ModelSim 6.4b
SPICE/IBIS Simulations	No
Implementation	
Synthesis Software Tools/Version Used	XST 12.3 (M70.D), Synplify_PRO E-2010.09-1
Implementation Software Tools/Versions Used	ISE® Design Suite 12.3
Static Timing Analysis Performed	Yes
Hardware Verification	
Hardware Verified	No
Hardware Platform Used for Verification	N/A

The device utilization data is shown in Table 6.

Table 6: **Device Utilization**

Parameter	Specification	
Configuration Parameter	POLY_LENGTH	31
	POLY_TAP	29
	INV_PATTERN	FALSE
Maximum Frequency (Using Slowest Speed Grade)	XC6VLX75T-FF484-1	500 MHz
	XC6SLX4-TQG144-2	270 MHz
	XC5VLX30-FF324-1	400 MHz
Device Utilization without Testbench (LUT6-Based Devices)	Registers	63
	LUTs	63
	Slices	16

References

1. Lin, S., and D. J. Costello. 2004. *Error Control Coding* (2nd Edition). Prentice Hall.
2. Peterson, W. W., and E. J. Weldon. 1972. *Error-Correcting Codes* (2nd Edition). The MIT Press.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
01/10/11	1.0	Initial Xilinx release.

Notice of Disclaimer