Application Note AN3101-12: Pseudorandom Numbers

by Shultz Wang

Introduction

In many applications, a source of random numbers is useful for processing purposes or as inputs. A white noise or pink noise source, for example, requires such a source. Since the DSP-1K is a digital machine, it cannot generate a purely random output. There are however several methods of calculating a value with a sufficiently random distribution such that it may be used to approximate a random process.

Algorithm

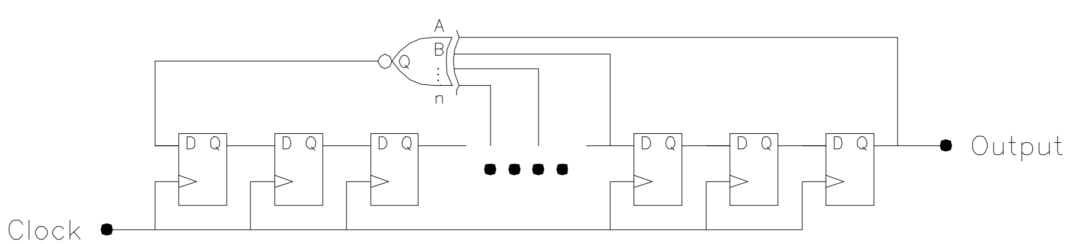
A common way to code a pseudorandom number generator (PRNG) is with a **linear congruential generator**, which follows the formula:

y[n] = (a*y[n-1] + c) modulo m.

Choosing the appropriate a, c, and m values is an art in itself. However, many references are available with tried and true choices. In this application, m is chosen to be 2^{16} to align with bit boundaries, a = 25173 and c = 13849 are good values to use with the selected m.

The code can take advantage of the non-saturation-limiting feature of integer operations in the DSP-1K for a modulo operation. However, this limits the number generated to 16-bits, giving a cycle time of 2^{16}/48000Hz = 1.37 seconds before bits are repeated. For higher bitwidths, the modulo operation must be explicitly coded.

A second way to code a PRNG is a **linear feedback shift register** (LFSR), a row of serially connected registers where the intermediate values are manipulated in a modulo-2 summation fashion to generate the next state. The implementation discusses in this application note uses the Fibonacci method, where the intermediate values are modulo-2 summed to generate the next input into the shift register.



Due to the lack of an XOR function in the DSP-1K, the modulo-2 summation is achieved by using a true summation with masking. A 25-bit LFSR is implemented in the example source code, giving a cycle time of 2^{25}/48000Hz = 699 seconds before bits are repeated. The LFSR generates one new bit every time it is executed, so depending on the number of pseudorandom bits needed in the application, the code may be duplicated multiple times, with the cycle time reduced accordingly.

In order to initiate the LFSR, the register where it is stored may need to be written with a seed value. If the logic performed in the modulo-2 math is equivalent to an XOR gate, then the LFSR must have a non-zero value in order to run, thus it requires a seed value which is not all 0's. If the logic performed in the modulo-2 math is equivalent to an XNOR gate, then the LFSR must have a non-max value in order to run, thus it requires a seed value which is not all 1's.

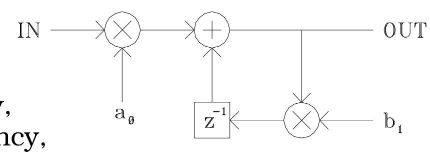
Application: Colored Noise

White noise (also known as Johnson noise, thermal noise, or shot noise) is defined as noise with equal energy density at every frequency, and can be used in frequency analysis of audio equipment, or for noise masking. The PRNGs generate a bitstream with a flat power spectrum distribution, and thus are perfect for white noise sources without further modification.

Pink noise (also known as 1/f noise, or flicker noise) is noise which has equal energy per octave, and thus falls off at a rate of -10dB per decade. This energy distribution is closer to what is found in nature than that of white noise, and is used for testing of speaker systems or room acoustics. With a filter to convert the spectrum of a white noise source to a 1/f distribution, a pink noise source can be created. Since a -10dB per decade filter has a gentler rolloff than even a single-pole filter, three filter are summed in this application note to approximate the response.

Brown noise is so named for the noise equivalent of Brownian motion, where the energy density falls off at a rate inversely proportional to the square of the frequency, or -20dB per decade. This is achieved by simply passing a white noise source through a single-pole filter, with the brown noise characteristics in effect above the corner frequency of the filter. Using a single-pole filter equation of:

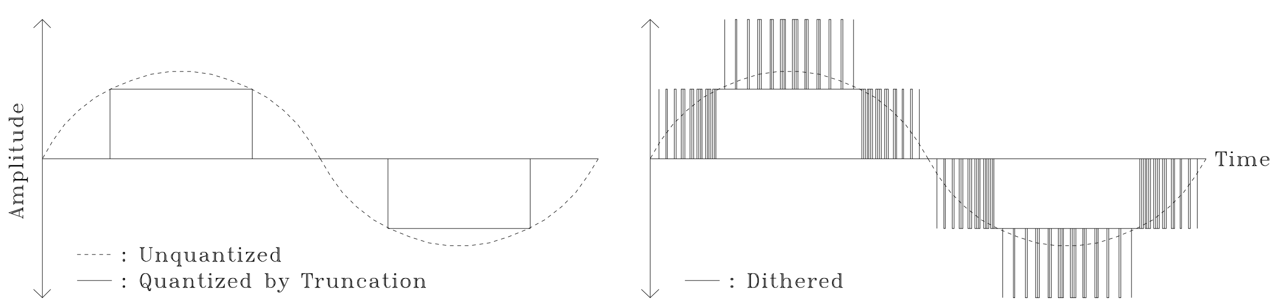
$y[n] = a_0 * x[n] + b_1 * y[n-1]$, where $a_0 = 1 - e^{-2\pi fc/fs}$,
$b_1 = e^{-2\pi fc/fs}$,
fc = corner frequency,
fs = sampling frequency,



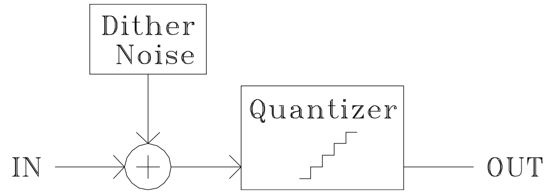
the active range of brown noise may be selected via the fc value. Setting fc = 1kHz, with fs = 48kHz, a_0 = 0.122694, b_1 = 0.877306.

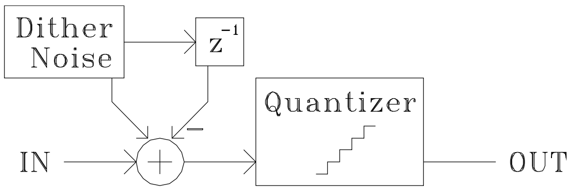
Application: Dither

Dithering is another application which requires pseudorandom numbers. Oftentimes the bitwidth of an output datastream has fewer bits than the bitwidth internal to the DSP, thus it is necessary to discard the extra bits through truncation, such as when the 24 bits of output from the DSP-1K has to be truncated to 16 bits in a particular application. This causes spectral and power correlations between the original signal and the quantization error. By adding a low amplitude white noise value to the original signal, the perceived stairstepping of the output can be minimized, and signals buried in the removed bits may be boosted to audible levels, at the expense of higher noise levels.

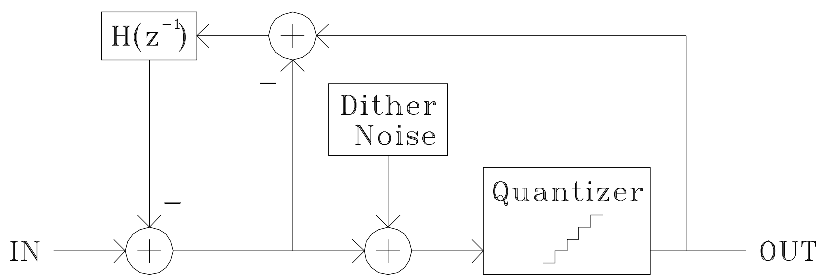


The most basic form of dithering is addition of a white noise source with an amplitude equal to the lowest unquantized bit to the pre-quantized signal, ±0.5LSB of the output. This is called **rectangular dither** due to its uniform probability distribution function, and has a noise penalty of 3dB.





To further decrease the modulation of the noise power by the input signal, a **triangular dither** may be used. This involves the combining of two dither sources, each having an amplitude of ±0.5LSB of the output, thus giving a triangular probability distribution function at ±1LSB of the output, and a noise penalty of 4.77dB. The combination may be done by summing two non-correlated noise sources, or by differencing two successive values from one noise source, which is in essence a 2-tap finite impulse response highpass filter. Since dither noise and quantization noise are additive, reducing audible noise is desirable. This filtering shifts the noise power towards higher frequencies, with a 50% reduction (3dB) in power at very low frequencies, and a 50% boost (1.76dB) at Nyquist. However, the white noise power spectrum does get replaced with a blue noise spectrum (rising energy with increasing frequency, the opposite of pink or brown noise), which some may find more objectionable.



The idea of filtering the noise to change its power spectrum can be taken one step further by applying **noise shaping**, which is the technique of using the negative feedback of the quantization error through a filter to shift most of the noise power to the higher frequencies, and thus take advantage of human psychoacoustic characteristics and reduce the apparent noise power levels. The effectiveness and resulting spectrum is highly dependent on the filter used. This application note presents several possibilities: a simple 1-stage delay:

$$H(z^{-1}) = z^{-1},$$

a second order FIR filter:

$$H(z^{-1}) = -2*z^{-1} + z^{-2},$$

and a eighth order Parks-McClellan optimized FIR filter:

fc = 18kHz, transition band = 1.92kHz,

$$H(z^{-1}) = 0.0399983 - 0.0786185*z^{-1} + 0.1268321*z^{-2} - 0.1652212*z^{-3} + 0.1798762*z^{-4} - 0.1652212*z^{-5} + 0.1268321*z^{-6} - 0.0786185*z^{-7} + 0.0399983*z^{-8}.$$

A more sophisticated filter can designed with an inverse A-weighted response in the lower frequencies, in order to more fully suppress audible noise levels.

Source Code

```
; File: AN3101-12LCG.ASM
; Description: Pseudorandom Numbers: Linear Congruential Generator
; Author: Shultz Wang
; Copyright: 2005 Wavefront Semiconductor

; DIRF: Pseudorandom number 'rnd' storage location
; y[n] = (a*y[n-1] + c) modulo m, a = 25173, c = 13849, m = 2^16

LCA    0      DIRF          ; Load y[n-1] into B
DAC    0      $189540       ; Load a into A
MLTB                        ; a*y[n-1]
XCM    0      DIRF          ; B=a*y[n-1]
DAC    0      $D8640        ; Load c into A
ADDB                        ; a*y[n-1] + c
SCA    1.0    DIRF          ; Write new rnd

CAD    0.0625 0.0           ; Scale +/-8.0 number to +/-0.5
SCA    1.0    OUT1          ; Output channel 1: White noise
```

```
; File: AN3101-12LFSR.ASM
; Description: Pseudorandom Numbers: Linear Feedback Shift Register
; Author: Shultz Wang
; Copyright: 2005 Wavefront Semiconductor

; DIRF: Pseudorandom number ‘rnd’ storage location
MEM    tmp    1      ; Dummy write location

;::::::: LFSR ::::::::
CM     $000001  DIRF ; Right-shift rnd 18 bits, bit 21 @ bit 3, bit 24 @ bit 6
SXCA   $001000  tmp  ; Right-shift rnd 18+6=24 bits, bit 24 @ bit 0
                     ; B = Right-shifted 18 bit rnd
SCBA   0.125  tmp    ; Right-shift rnd 18+3=21 bits, bit 21 @ bit 0
                     ; (Bit 21 @ bit 0) + (bit 24 @ bit 0) = bit 21 XOR bit 24 @ bit 0
1AC    $1            ; (Bit 21 XOR bit 24 @ bit 0) + 1 = bit 21 XNOR bit 24 @ bit 0
                     ;   ** This instruction changes the requirement of the seed value
                     ;   from a non-zero value to a value that is not all 1’s
ANDC   $1            ; Screen out bit 0
CMA    2.0    DIRF   ; Put new bit 0 into rnd
ANDC   $1FFFFFF      ; Remove bit 26 from rnd
SCA    1.0    DIRF   ; Write new rnd
  ;* Repeat above code by the number of bits needed in pseudorandom number, up to 25x

;:: Sign extension :::
ANDC   $1000000      ; Isolate bit 25
SKIP   Z      fi     ; If (bit 25 != 0)
CM     -1.0   DIRF   ;   2's complement rnd
ANDC   $1FFFFFF      ;   Remove sign bits
CAD    -1.0   0.0    ;   2's complement (2's complement rnd) = inverted sign bits
SKIP          esle   ; Else
fi:
CM     1.0    DIRF   ;   Load rnd
esle:
CAD    0.25   0.0    ; Scale 25 bit number to 23 bit

SCA    1.0    OUT1   ; Output channel 1: White noise
```

For **pink noise**, replace the last line of the PRNG code with the following filter.

```
; DIRC: y0[n] = a00*x[n] + b10*y0[n-1], a00 = 0.0990460, b10 = 0.99765
; DIRD: y1[n] = a01*x[n] + b11*y1[n-1], a01 = 0.2965164, b11 = 0.96300
; DIRE: y2[n] = a02*x[n] + b12*y2[n-1], a02 = 1.0526913, b12 = 0.57000
; Pink = y0[n] + y1[n] + y2[n] + g*x[n], g = 0.1848
SXCA   0.0990460    tmp    ; a00*x[n], B=white
CMA    0.99765      DIRC   ; a00*x[n] + b10*y0[n-1]
SCB    0.2965164    DIRC   ; a01*x[n], y0[n] = a00*x[n] + b10*y0[n-1]
CMA    0.96300      DIRD   ; a01*x[n] + b11*y1[n-1]
SCB    1.0526913    DIRD   ; a02*x[n], y1[n] = a01*x[n] + b11*y1[n-1]
CMA    0.57000      DIRE   ; a02*x[n] + b12*y2[n-1]
SCB    0.1848       DIRE   ; g*x[n], y2[n] = a02*x[n] + b12*y2[n-1]
CMA    1.0    DIRE   ; y2[n] + g*x[n]
CMA    1.0    DIRD   ; y1[n] + y2[n] + g*x[n]
CMA    1.0    DIRC   ; Pink = y0[n] + y1[n] + y2[n] + g*x[n]

SCA    1.0    OUT1   ; Output channel 1: Pink noise
```

For **brown noise**, replace the last line of the PRNG code with the following filter.

```
; DIRB: LPF
; Brown = y[n] = a0*x[n] + b1*y[n-1]
;    For fc = 1kHz, fs = 48kHz: a0=0.122694, y1[n]=0.877306
CAD    $7DA4  0.0    ; a0*x[n]
CMA    $3825C DIRB   ; a0*x[n] + b1*y[n-1]
SCA    1.0    DIRB   ; Save new y[n]

SCA    1.0    OUT1   ; Output channel 1: Brown noise
```

For **rectangular dither**, replace the last line of the PRNG code with the following code.

```
; Rectangular dither
CAD    $4     0.0           ; Scale number to below 16th fractional bit
CMA    1.0    IN1           ; Input plus dither
ANDC   $FFFFF00             ; Truncate to 16 bit output

SCA    1.0    OUT1          ; Output channel 1: Rectangular dithered input
```

For **triangular dither**, store a copy of the PRN from the last tick in register B, and replace the last line of the PRNG code with the following code.

```
; Rectangular dither
; Triangular dither
MEM    tmp    1             ; Dummy write location
SCBA   -0.0625      tmp     ; y[n] - y[n-1]
CAD    $8     0.0           ; Scale number to below 15th fractional bit
CMA    1.0    IN1           ; Input plus dither
ANDC   $FFFFF00             ; Truncate to 16 bit output

SCA    1.0    OUT1          ; Output channel 1: Triangular dithered input
```

For **dither with noise shaping**, replace the last line of the PRNG code with the following code.

```
; Dither with noise shaping
MEM    FIR    $n            ; n-tap FIR filter buffer, replace with actual tap count
MEM    Stor   1             ; Pre-dither storage

CAD    $8     0.0           ; Scale number to below 15th fractional bit
... Filter code to be placed here ...
CAM    -1.0   IN1           ; Input minus filtered error
SCAB   1.0    Stor          ; Add dither, Stor = Pre-dither
ANDC   $FFFFF00             ; Truncate to 16 bit output

SCA    1.0    OUT1          ; Output channel 1: Noise shaped dithered input

CMA    -1.0   Stor          ; Post-dither - Pre-dither = Error signal
SCA    1.0    FIR           ; Put error signal into FIR filter buffer
```

The **first order filter**, consisting of a single delay, is only one line of code.

```
XCM    1.0    FIR+1         ; 1st tap times coeff, B = dither
```

The **second order filter** is as follows.

```
XCM    -2.0   FIR+1         ; 1st tap times coeff, B = dither
CMA    1.0    FIR+2         ; 2nd tap times coeff
```

The **eighth order filter** is as follows.

```
XCM    -0.0786185   FIR+1  ; 1st tap times coeff, B = dither
CMA    0.1268321    FIR+2  ; 2nd tap times coeff
CMA    -0.1652212   FIR+3  ; 3rd tap times coeff
CMA    0.1798762    FIR+4  ; 4th tap times coeff
CMA    -0.1652212   FIR+5  ; 5th tap times coeff
CMA    0.1268321    FIR+6  ; 6th tap times coeff
CMA    -0.0786185   FIR+7  ; 7th tap times coeff
CMA    0.0399983    FIR+8  ; 8th tap times coeff
1AC    0.0399983           ; FIR offset
```

NOTICE

Contact Information:

Wavefront Semiconductor
200 Scenic View Drive
Cumberland, RI 02864 U.S.A.
Tel: +1 401 658-3670
Fax: +1 401 658-3680
On the web at www.wavefrontsemi.com
Email: info@wavefrontsemi.com

Application note revised March, 2005