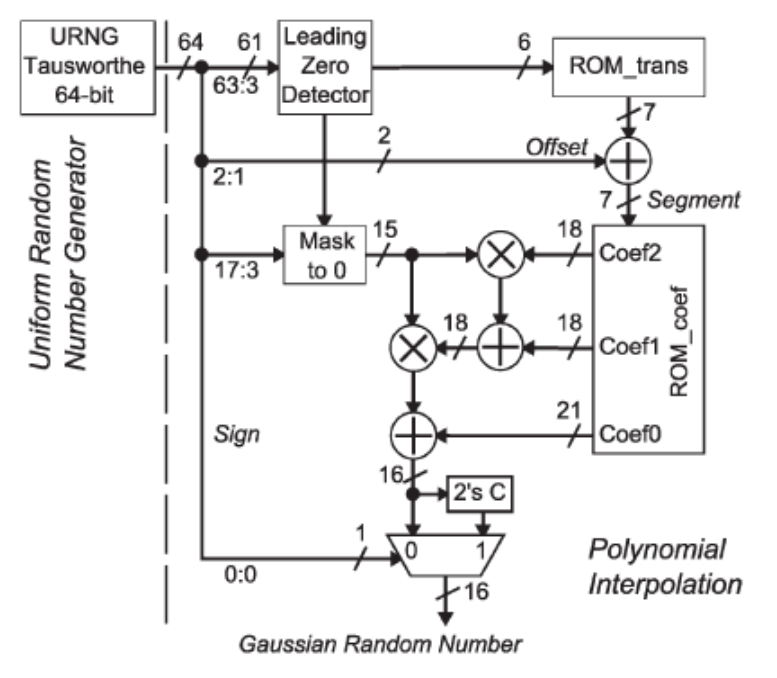
# Gaussian Noise Generator Design Description Document

## Background

This module implements a Gaussian distributed, random number generator based on the paper “Hardware Architecture of a Gaussian Noise Generator Based on the Inversion Method” by Gutierrez, *et al.* The design consists of a 64-bit Tausworthe uniform random number generator and a piece-wise polynomial approximation of the inverse cumulative distribution function (ICDF).



The design is archived with a Matlab reference source and a Verilog-based Vivado project.

## Toolset

The design was implemented using the following toolset:

* Matlab R2016b (with Symbolic Math toolbox for coefficient calculation)
* Vivado 2016.3 WebPack edition

## Matlab Design

The Matlab implementation is executed by running the *gng\_test.m* script in the Matlab folder. This script will generate 10000 random samples, storing the output and intermediate values in the ref\_results folder for use by the Verilog testbench.

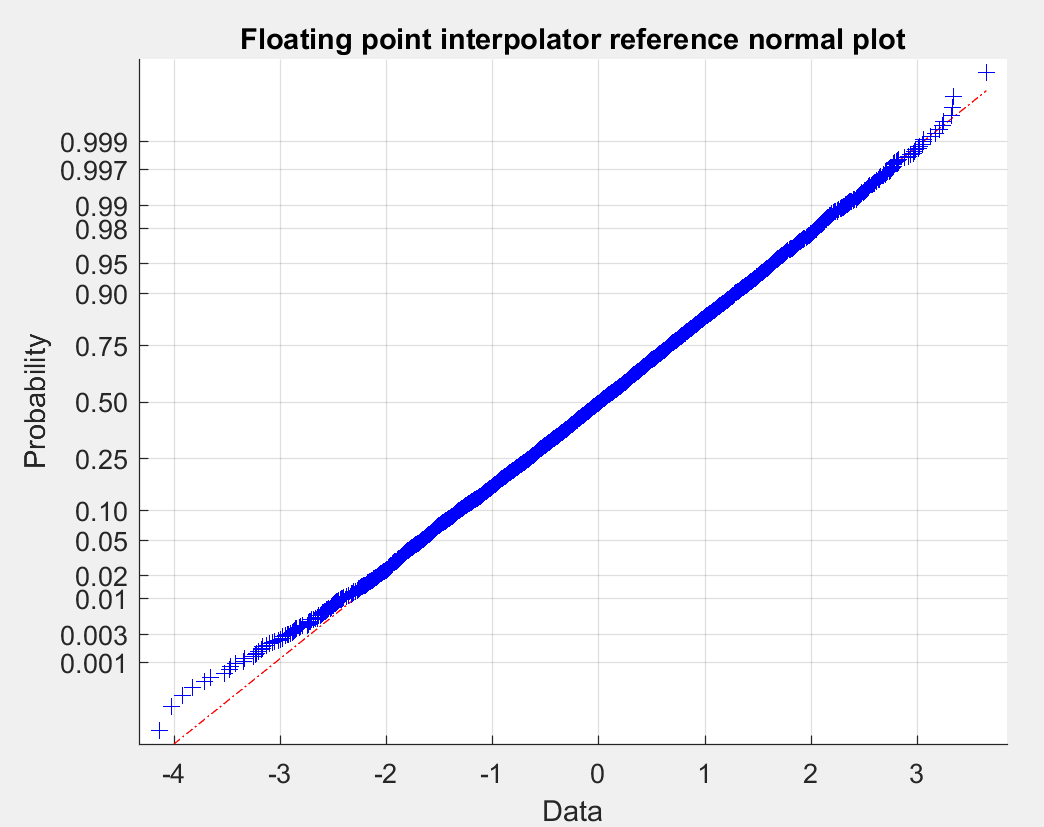
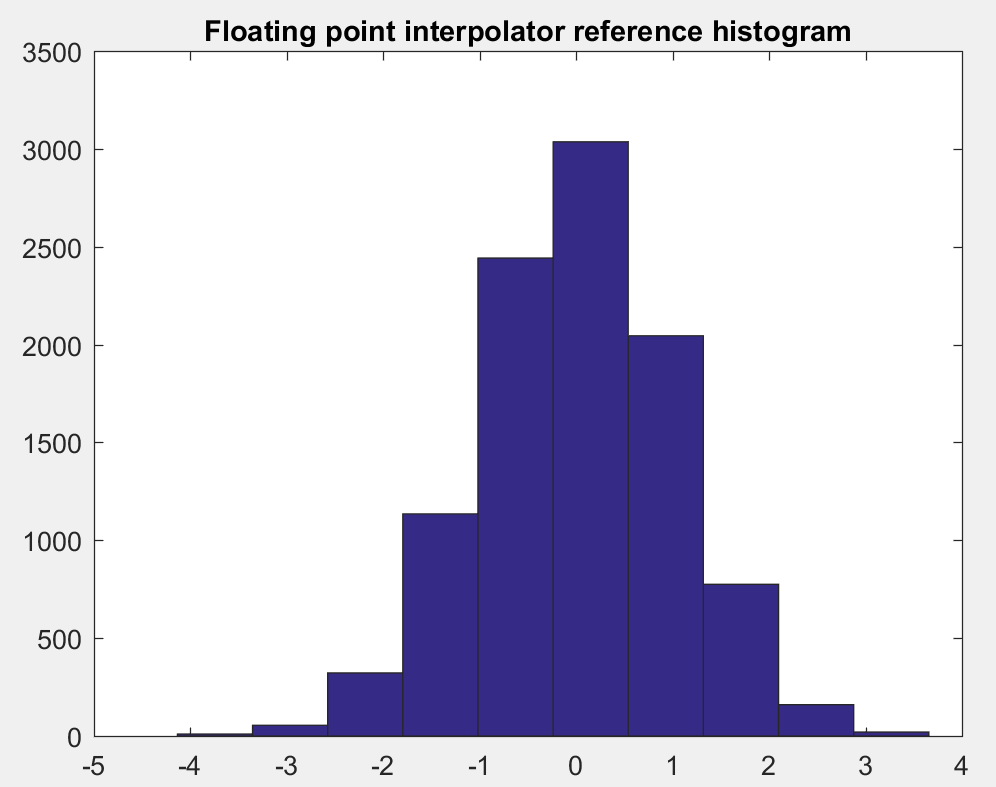
The design achieves the following statistical results:

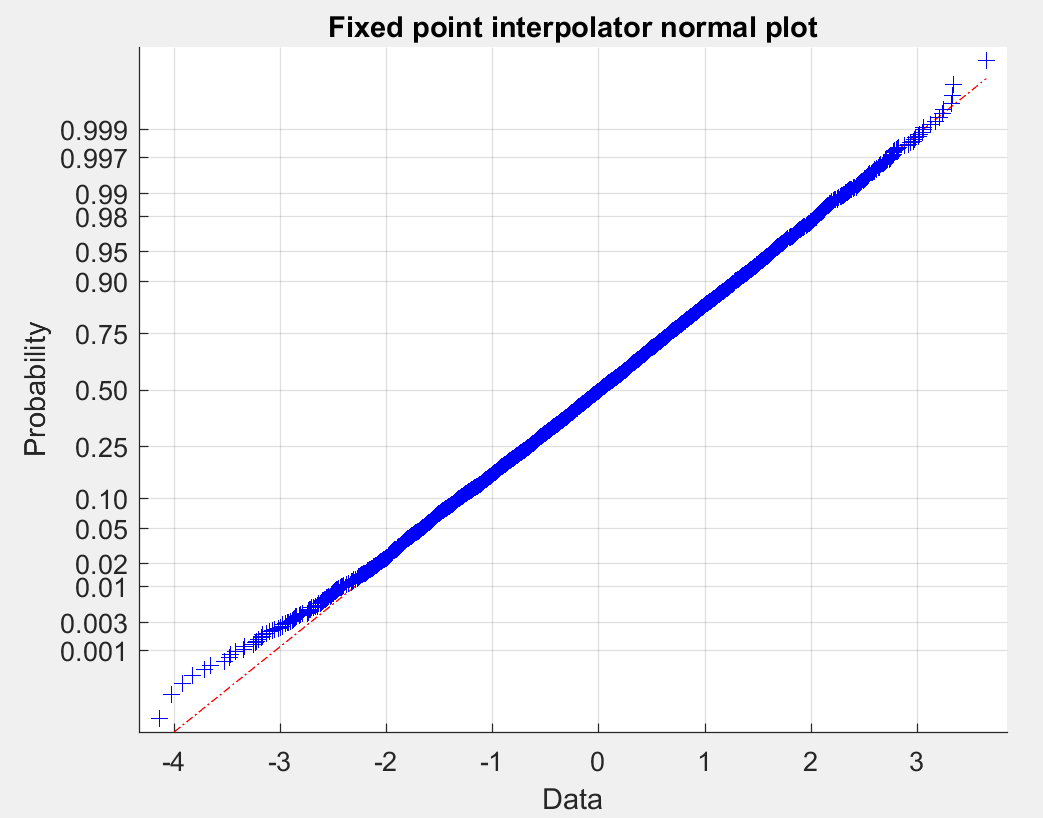
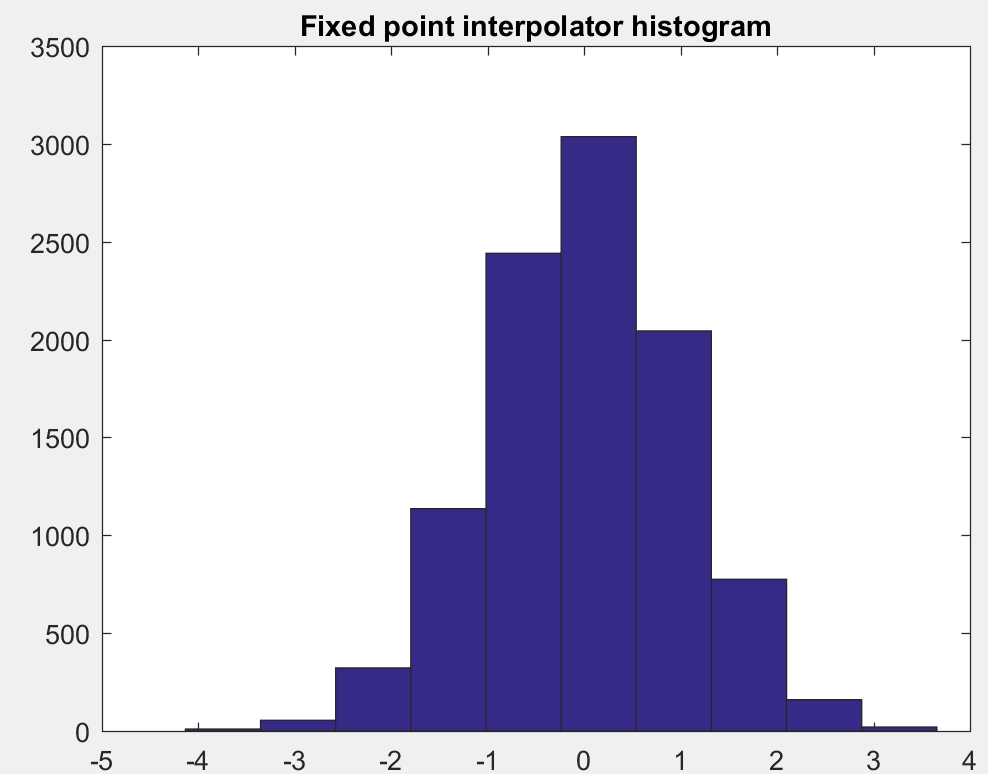
Floating point interpolator ref results:

H:0, P:0.16358, KS Stat:0.01117, CV:0.013564

Fixed point interpolator results:

H:0, P:0.16583, KS Stat:0.01114, CV:0.013564





### Coefficient Calculation

For this implementation, the ICDF is approximated by polynomial interpolation. As the ICDF is symmetric about 0.5, the coefficients are calculated only for the region 0.5 to 1 in order to only use the positive region. As the ICDF becomes increasingly non-linear as it tends to 1, the input range is divided using a hierarchical segmentation scheme, determined by the number of defined inner and outer bins.

The outer segmentation works as a power-of-2 segregator (i.e. the first outer bin covers the region 0.5 to 0.75, the second 0.75 to 0.875, and so on) while the inner segmentation is a uniform segregation. The *coeff\_calc\_P2S\_US.m* function loops through all outer and inner bins, performing a Matlab *polyfit* for the partial region of the ICDF of the active bin against an input of 0 to 1 in 2^-15 steps.

Due to the numerical precision required when using more than ~35 outer bins, this function is implemented using the Symbolic Math toolbox. This results in very long run times so a reference *coeffs.mat* and *fix\_coeffs.mat* are provided with floating- and fixed-point coefficients respectively.

### Uniform Random Number Generation

The required uniform random numbers are generated by a 64-bit Tausworthe generator. Note that the initial values of the intermediate registers need to match the values used in the Verilog implementation in order to obtain the same results.

### Leading Zero Detection

The *lzd.m* function identifies the position of the most significant 0 in a 64-bit binary number.

### Mask to 0

The *mask\_to\_zero.m* slices bits 17 down to 3 from a 64-bit input number, bit reverses it, and masks to zero any LS bits that came before the most significant 0 identified by the Leading Zero Detector. The output of this function is treated as a ufix(15,15).

### Interpolation

Both a floating- and fixed-point version of the interpolator are provided in *gng\_test.m*. Note that the fixed-point scalings are based on the fixed-point requirements of the coefficient table calculated using 62 outer segments and 4 inner segments. If values other than this are used, the scalings would need recalculated.

### Data Logging

At the end of the test, the output and intermediate fixed-point results are saved as text files in the ref\_results folder. These text files are used by the Verilog testbench to verify a bit-accurate implementation has been achieved.

## Verilog Design

The top level Verilog module, *gng\_top.v*, instantiates a 64-bit Tausworthe uniform random number generator, *urn\_gen.v*, and an ICDF interpolator, *gng\_interpolator.v*.

### Uniform Random Number Generation

The module *urn\_gen.v* implements a 64-bit Tausworthe generator. This module resets the intermediate registers to a known starting starting condition when the *rst* input is asserted. When *rst* is de-asserted, a new random number is generated at every rising edge of *clk*.

### Interpolation

The module *gng\_interpolator.v* implements the ICDF interpolator. This module instantiates the leading zero detector sub-module (*lzd.v*), mask-to-zero sub-module (*mask\_to\_zero.v*), the coefficient store (*coeff\_store.v*), and two multiply-add modules to calculate the polynomial approximation.

Various bit-shifts are implemented to ensure binary point alignment and various shit register pipelines are implemented to ensure latencies are balanced through the design.

## Testbench Description

The Verilog implementation is verified against the Matlab fixed-point implementation by the *gng\_top\_tb.v*. This testbench instantiates *gng\_top* and compares its output to the expected results from the Matlab-generated text files.

A shift register (*test\_en\_reg*) is used to enable the individual tests at the appropriate time based on the internal latencies of the design.

If any differences are found between the implemented results and the expected results, the testbench raises a warning and describes which sub-module has caused the error. When all stored reference values have been tested, the testbench terminates and issues an overall pass/fail message.

Note that if any of the fixed-point scalings or the initialization values for the Tausworthe generators are changed, resulting in changed reference result text files, the new text files need manually add to the Vivado project by adding them as simulation sources (ensure that “Copy sources into project” is selected).

## Implementation Results

The design was synthesized and implemented for a Kintex Ultrascale xcku035-ffva1156-3-e. The achieved results were:

+----------+------+

| Resource | Used |

+----------+------+

| LUT | 331 |

| LUTRAM | 22 |

| FF | 396 |

| BRAM | 1 |

| DSP | 2 |

| IO | 18 |

| BUFG | 1 |

+----------+------+

| FMax | 583M |

| Thruput | 583M |

+----------+------+

Sample timing constraints are provided as part of the Vivado project. Note that I/O pins have not been constrained as this core has not been targeted to a specific board.

## Future Work

This design has been implemented using a fixed segmentation scheme which gives adequate but potentially non-optimal partitioning of the inverse cumulative distribution function. A future enhancement could be to implement a segmentation scheme such as the proposal in “Hierarchical Segmentation for Function Evaluation” by Lee, *et al.*, which could result in a reduced memory requirement and increased tail accuracy.