Gaussian Noise Generation using Box-Muller Method

Features:

- ✓ Designed a synthesizable Gaussian white noise generator using Box-Muller method in Verilog.
- ✓ Time period of the noise samples is 2^{88} which almost equal 10^{25}.
- ✓ Noise generated is quantized to 16 bits with 5 integer bits and 11 fractional bits.
- ✓ Bit accurate Matlab programs for seed and noise programs included.
- ✓ Five stage pipelined architecture. Produces 2 noise signals for every 1 clock cycle.

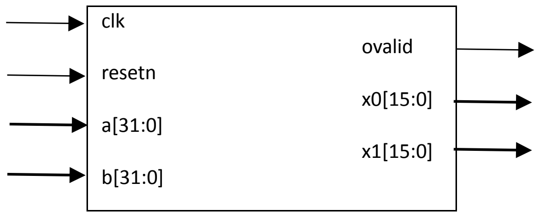
Tools and Software used for design:

HDL Used	Verilog
Simulation	Modelsim 10.4b
Verification	Matlab R2015b + Modelsim 10.4b.

Pin Description:

Pin Name	Direction	Functionality
clk	Input	Triggers the sequential circuit for every rising edge.
resetn	Input	Active low synchronous signal that resets the complete circuit
a	Input	Seed0
b	Input	Seed1
x0	Output	Gaussian Noise output0
x1	Output	Gaussian Noise output1
ovalid	Output	High when the output noise is valid

Circuit Diagram:



Approach:

The problem statement was approached in a formal way by closely following the real time ASIC design flow. Below are these steps followed in the last two weeks:

- ✓ *Design Specification:*
  - The given requirements were modularized to understand the complexity of the problem.
  - After thorough analysis, it was concluded that the multipliers, the look up tables for the polynomial coefficients, leading zero detectors form the critical path of the design.
  - Detailed analysis was done to arrive at the bit widths of intermediate signals to get more precision and accuracy.

- ✓ *Behavioral Description:*
  - The second step was to arrive at the behavior of each submodule. Requirements of each submodule were listed.
  - Based on the requirements of the submodules, the logic of the system is developed.
  - Timing diagrams for the pipelined architecture were drawn to understand the data path. This step has significantly reduced the effort during the RTL coding.

- ✓ *RTL Coding:*
  - The designed logic was converted into RTL using Verilog.
  - Special care was taken to ensure that the logic implemented is synthesizable.

- ✓ *Simulation:*
  - After the implementation of each submodule, simulation was performed with various test vectors to ensure its proper functionality.

- ✓ *Integration:*
  - Integrating all the submodules was the most time consuming part of the project.
  - Certain mistakes during the incremental testing were identified and more time was spent to rectify them.

Implementation:

- ✓ *Pipelining:*
  - The IP core was designed using Verilog. After arriving at the specifications, it was concluded that pipeline architecture can be used to achieve higher performance.
  - It was evident that there is inherent parallelism in the calculations of cos/sine and logarithm along with square root form independent data path. For this reason, the processing of these functionalities are designed to occur in parallel.
  - In each of these parallel paths, the independent steps were clubbed together to form a pipelined architecture. It was observed that the algorithm can be divided into 4 stage pipeline.
  - However, the cos/sine functions can be calculated a clock cycle before the calculation of the logarithm and square root functions. Therefore, a clock cycle delay is implemented to achieve the required timing of the signals.

- ✓ *Polynomial Coefficients:*
  - ▪ The coefficients of logarithm, square root and cosine functions are generated using "polyfit" command in matlab.
  - ▪ The input interval is split into required segments and coefficients for each segment are generated. Since the range reduction and reconstruction steps were followed in each step, the coefficients were adjusted accordingly.
- ✓ *Multipliers:*
  - ▪ The multipliers form a critical path in this implementation. So the possible options were to pipeline operation or to design them in combinatorially. However, the latter option was chosen as the main aim was to simulate and not synthesis.
  - ▪ The precision of the outputs of the multipliers were taken care at each step. Faithful rounding of the outputs was performed at each step.
- ✓ *Leading Zero Detector (LZD):*
  - ▪ The Leading Zero Detector (LZD) circuit was implemented in the way proposed in [2]. The 2 bit LZD is basic building block. A 4, 8,16,32,48 bit LRZ were constructed using the 2 bit LRZ.

Challenges Faced:

The major challenges involved in this project are listed below:

- ✓ Since Verilog is strongly inclined towards the unsigned numbers, more effort was required to verify the representation of negative numbers in their corresponding two's complement notation.
- ✓ The multipliers demanded more effort, as the errors (quantization or logical errors) affected the outputs significantly.
- ✓ 30% of time was spent on integration of the submodules. Several unexpected issues made the integration challenging.

Design Verification:

The design verification is done by comparing the golden matlab output with the RTL output every clock cycle. However, after the design and implementation of the given specifications, I arrived at the given output. I understand that in the given time frame, I was unable to achieve the expected output.

The following areas in the code are suspected:

- ▪ In the calculation of the logarithm, the output of the logarithm function is supposed to be [0,66.54]. Incase this calculation goes wrong, the error is propagated throughout and leads to undesirable outputs.
- ▪ In the square root calculation, it is understood that the input interval is in [1,4). The interval is split into two sub intervals i.e [1,2) and [2,4). Each of these segments are divided into 64 segments. It is obvious that the interval [2,4) would not provide accurate results, as the segment width is twice that of the segment width in [1,2) interval.
- ▪ Also the leading zeros should be calculated for the input of the square root function. If the input is greater than zero, the functionality works fine. However, if the input is less than zero, malfunction occurs.

I am sure that the error is located in the above mentioned areas. I am confident that I could arrive at the expected output in an extended time frame. However, it is due to the fact that I had not foreseen the integration issues and this could the major reason for incorrect final output.