|  |
| --- |
| Santa Clara University |
| Implementation of AWGN |
| Using the Box-Muller Method |
|  |
| **Kevin Childs** |
| **3/12/2016** |

|  |
| --- |
| This paper discusses some of the trials and tribulations in building a “hardware” version of an Additive White Gaussian Noise (AWGN) generator. I begin by discussing complications in System Generator and then go into some of the algorithms used in my Matlab simulation. I then show the results of this implementation. |

# Introduction

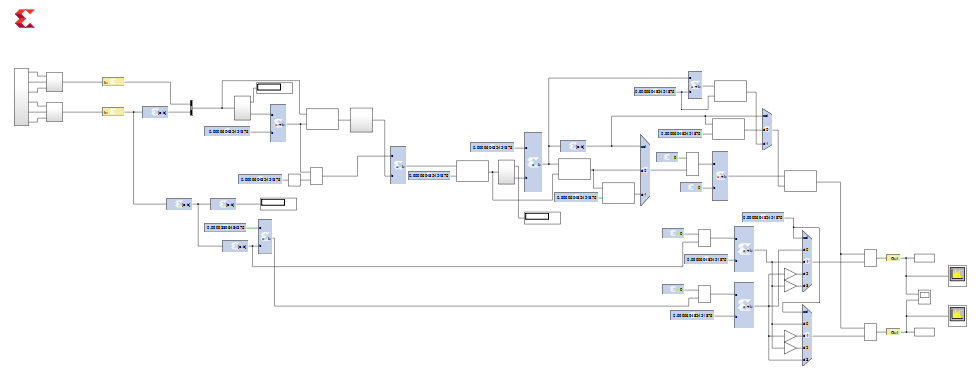
Additive White Gaussian Noise (AWGN) is invaluable to signal processing. Not only can it be used to knock down spurious signals from constructive interference in FPGA signals, but it is at the core of LPC Speech Coding reproduction. With that said, it is necessary to ensure that this noise is as random as possible, so that a “full” reproduction of speech can be obtained and constructive interference can be mitigated. With the computational demand of many random generating algorithms, implementation of these designs can be utilized on chipsets to speed them up. These ideas are outlined in [1] more fully. In our project, we were to implement this design and create the RTL’s for it. Although I was able to get a simulation in Matlab created, due to my limited knowledge of Vivado and lack of time after failing with System Generator, my hardware construction was an impossibility. Hopefully, by seeing what road blocks I was up against and how I overcame most of the obstacles I overcame, it will be noticeable the efforts I put into this endeavor and the knowledge I gained.

# Initial Design Attempt Using System Generator

When I started this project, my natural intuition was to use a program that I knew had a better likelihood of accelerated prototyping from design to hardware implementation. That is why I chose to work with System Generator and the Vivado Suite. System Generator uses a set of modularized blocks much like Simulink, but designed for straight implementation into simulated design and quick synthesis on an FPGA. My approach was to get a full simulation created of the System Generator design and then synthesis would be no more than creating the RTLs for a particular chipset and then synthesizing it to an FPGA board. Although this sounds straight forward, System Generators error messages aren’t exactly great indicators of the problem, and having no more than perhaps 100 hours of practice using System Generator from my ELEN226, FPGA design class, I was by no means an expert. Therefore, after dozens of hours troubleshooting and even going to the extent of a teleconference with the Professor, I decided it was in my best interest to throw in the proverbial towel for this design. The following sections illustrate some of the design and errors that I received.

## System Generator Design

Below you can see the entire implementation of System Generator:



Although this design did not implement all the correct cordics and was by no means free of errors, getting even a more simple simulation performed was a difficult task. Below are a few of the methods I tried to use just for the Tausworthe URNG that ended in failure:

### S-Function Module:

|  |  |
| --- | --- |
| S-Function | S-Function Error |

### Black Box Module:

|  |  |
| --- | --- |
| Black Box Error – Could not find file (file highlighted on left) | Black Box Error– unable to initialize module |

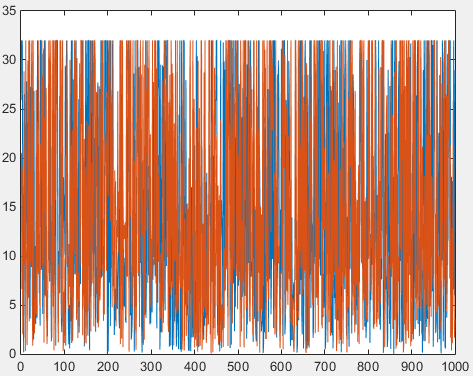
Without being able to create the Tausworthe UNRG or simulate due to an assortment of errors in my model, I decided to stop while I was behind. Therefore, I started to code using Matlab.

# Later Design Attempt Using Matlab

Although I was behind the curve with only a week to go, I felt more natural in the simplistic Matlab coding environment. Even though it was going to take learning more code and longer searches on the internet to get a feel for the fixted point implementation within Matlab, this methodology allowed for a more “divide and conquer” approach by allowing more coder friendly troubleshooting techniques (not all variable needed to be tied off as they did in System Generator). So with that said, all I needed to do was follow a bit stream through all the methods of the paper [1]. Using Figure 6 as a model and Figure 2 to get the necessary bit lengths, I was able to get a working model within roughly 20 hours of coding. The reason it wasn’t quicker, was because I needed to become familiar with the fixed point representation of the code. By using “fi”, I was able to make quick work of Figure 6 with the properly quantized values throughout the process using Figure 2. This was no easy task though; I was required to make sure no bit overflow occurred in the process, and I am not fully convinced I was able to accomplish this. I also used programming techniques rather than all inclusive hardware design implementations. This means that creation of the RTL file would not be possible until these design bugs were fixed. Although the Leading Zero Detector (LZD) hardware design was provided in the paper [1], my implementation of it was not without bugs, and this late in the game, I needed to get something working or I would feel that I accomplished nothing for the amount of time I spent on this. Thus, I just walked through the array looking for the first one and returned the array value.

# Results

Due to the inability to simulate my System Generator version of this design, I was unable to get any results to see if my design was viable; but, as for my Matlab code, I was able to get results close to Figure 12 of the paper[1].



As you can see from the plot. It appears that there is some overflow from the truncation of the tops of these signals. I notice this earlier in the code and was able to alleviate it by using more bits during calculations and then shifting it to fit within the recommended bit length, but with the outputs being set at 16 bits with 11 fractional bits, the prior calculation must have overflowed rather than truncated / rounded. This is another issue I would have worked on given the time.

# Conclusion

Due to the time constraints and the difficulty of the project, I am quite pleased with the results I achieved. I had some major hurdles in this project, not being from this discipline of studies and not having my hands into some research of this type. My over-optimism with thinking that I could simulate and synthesize in the System Generator / Vivado environment left me with little time to regroup, but with only four days to redesign this implementation, I was able to get this code working, cordics and all. If I had to do it all over again, I would definitely start with simple coding and if time permitted, I would go to Simulink / System Generator implementation or directly to Vivado. Although I didn’t get the required RTLs that my PhD candidate peers did, I would say that I achieved a lot for being one of only two Master’s students in this class. I was quite happy with the results, but wish I had more time to follow this through to completion.

# References

1. Dong-U Lee, John D. Willasenor, Wayne Luk, and Philip H. W. Leong, “A Hardware Gaussian Noise Generator Using the Box-Muller Method and Its Error Analysis”, “http://ieeexplore.ieee.org/xpl/login.jsp?reload=true&tp=&arnumber=162895”, IEEE.