Tutorial - Noise Source

From Casper

Jump to: navigation, search

Tutorial 6: Study and Implementation of Digital Noise Source

Author: Kaushal Buch (Version 1)

Expected completion time: 2 hours

Contents

[hide]

Hardware and software used for this tutorial

Hardware/Software	Description
PC	Dell Intel(R) Core(TM) i3 CPU 530 @ 2.93GHz width

	64 bit & 4GB RAM
OS	Linux 2.6.35-30-generic #54-Ubuntu 10.10 SMP x86_64 GNU/Linux
Matlab	2008a
Xilinx	ISE version 11.5
CASPER lib	gits_100511
Python	version 2.6
corr package	corr-0.6.5
minicom	version 2.4 (compiled on Jun 3 2010)
ROACH unit	version 1.0 Rev 3 2009 , uboot : uboot-2010-07-15-r3231-dram , Linux Kernel Image : uImage-jiffy-20091110

Introduction

In this tutorial, you will study the concept of the digital noise source and implement it using the Simulink-System Generator based tool-flow & CASPER library. You will download the BORPH (.bof) executable file on ROACH and will view the noise histogram on a P.C. by running a python script.

Setup

The lab at the workshop is preconfigured with the CASPER libraries, Matlab and Xilinx tools. Please refer the file LOCATIONSandFILES in the home/Desktop area or LOCATIONSandFILES slides displayed, for the locations/directories and files information required in the tutorial.

Note: The Date and Time portion of the BOF file name will be different! It depends upon when (Date & Time) you complile your model file !

Note: All the following cable connections and entries in the /etc/* files of the workshop PCs have already been done. You are not required to do any of the following setup and they are informatory in nature. You can verify points 1 to 4 on the setup you are working on and if you have any doubts regarding them kindly contact the lab instructor. Kindly go through point 4 to decide the way you will implement the tutorial.

1. Connect the Serial port cable between the ROACH board's P2 connector and serial port of the PC (on which minicom program exists).

2. Connect the Ethernet cable to J25 port of the ROACH board from the PCs eth1 port. /etc/ethers file should have mac address and corresponding IP address. In the /etc/network/interfaces file , eth1 should be configured. And in the file /etc/hosts , IP address and corresponding roach board (host) name entry to be done.

3. seed_value.m file must be kept in the directory [MATLAB_START_DIR].

4. Create your own directory at “[USER_DIR]” , where you can save and compile your model file or save any work that you may do. There are three ways to implement this tutorial.

A)You can either copy the mdl file “[TUT6_MDL_FILE]” from the the area “[STD_MDL_DIR]” to the directory that you have created at “[USER_DIR]” and compile it in the MSSGE (Matlab-Simulink-System Generator) environment

OR

B)You can use the bof file kept in the area “[FPGA_PROG_BOF_DIR]/[TUT6_BOF_FILE]” to directly program (using the python script explained in “Running the design”) the FPGA and look at the results

OR

C)Follow the steps given below to create the mdl file similar to the file “[STD_MDL_DIR]/[TUT6_MDL_FILE]”.

5. Start the matlab :

$cd [MATLAB_START_DIR]

[MATLAB_START_DIR]$./[MATLAB_START_FILE]

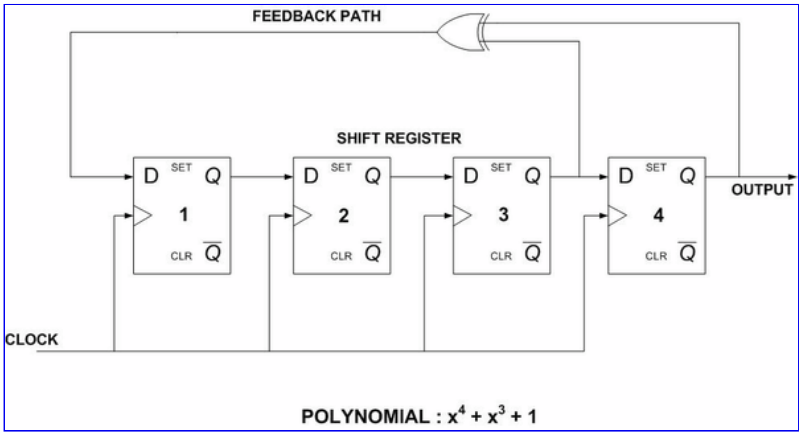
Design of Digital Noise Source

Noise sources are required for the testing of hardware in absence of an actual input or in general to emulate the behaviour of noise. During the development of digital receiver design for radio telescopes, noise source is required for testing or, one has to check it, taking the output from the antenna (i.e. the actual operational receiver chain). The aim of this tutorial is to study and implement a digital noise source using the CASPER tool flow. A digital noise source can be implemented as a part of the receiver hardware which can be used to provide noise stimulus for verifying the digital design (correlator) on FPGA.

Various methods exist for generating Gaussian random numbers on hardware. We would be considering the use of Central Limit Theorem to generate Gaussian random numbers (noise) from random numbers with uniform probability distribution. The hardware requirement is relatively lesser for this approach and also it does not use any FPGA specific hardware like BRAM, multipliers etc.

Central Limit Theorem (CLT) states that when finite variance random variables having different phases are combined, it leads to random variable possessing a Gaussian distribution.

Uniform random number can be generated digitally using a Linear Feedback Shift Register. A maximal length LFSR can generate uniform random variables with a periodicity of $2^n - 1$, where n is the order of the polynomial. An example of $x^4 + x^3 + 1$ polynomial using LFSR is shown in the figure below -



The polynomial has to be chosen from the set of maximal length sequences, similar to ones used for generation of PN sequences in CDMA transmitter. Irreducible polynomials of degree L generate sequences whose periods must be divisors of $q^L - 1$. Those special irreducible polynomials, which are the characteristic polynomials of m-sequences, are called primitive polynomials, and they exist for every degree over every finite field.

A variant of the normal serial output LFSR known as the Leap Forward LFSR is used. The leap forward LFSR would generate a parallel output, by 'looking-ahead' number of bits required by the output vector. In our case, 14 LFSRs each having a periodicity of 2^{18}-1 values, each of 4-bit are chosen for implementation.

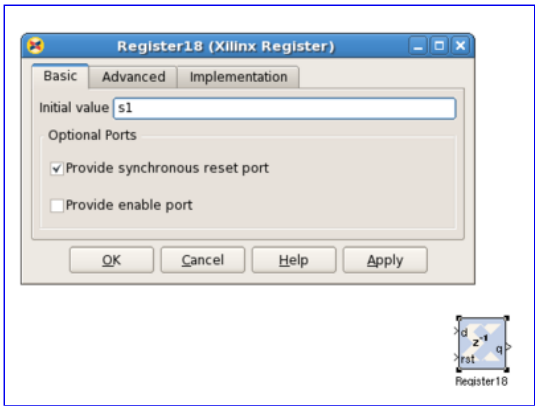
Creating LFSR Design

Create a New Model

Start Matlab and open Simulink (either by typing simulink on the Matlab command line, or by click in the Simulink icon in the taskbar). Open a new .mdl file and start the design with an LFSR.

Adding a register

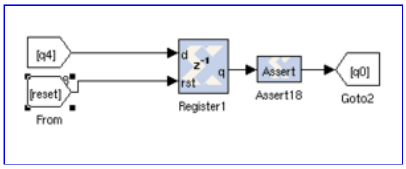
Add a register from the library. Double click the register and select the synchronous reset port option. In the field of initial value put s1.



Duplicate the registers and add Assert Block

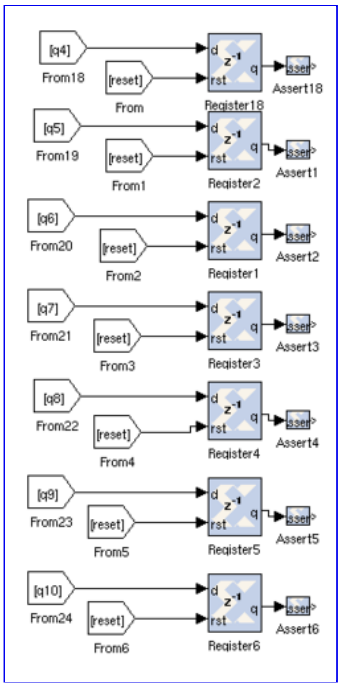
As we are designing for an 18-bit polynomial, we would require 18 registers. Duplicate all the registers from the first one, double click and add the Initial value from s2 to s18 respectively in all the registers.

The LFSR structure needs a feedback and Simulink requires an assert block when the output is fed back to the input. Connect the 'Assert' block at the output of the register and double click and set the output data type to unsigned Boolean.



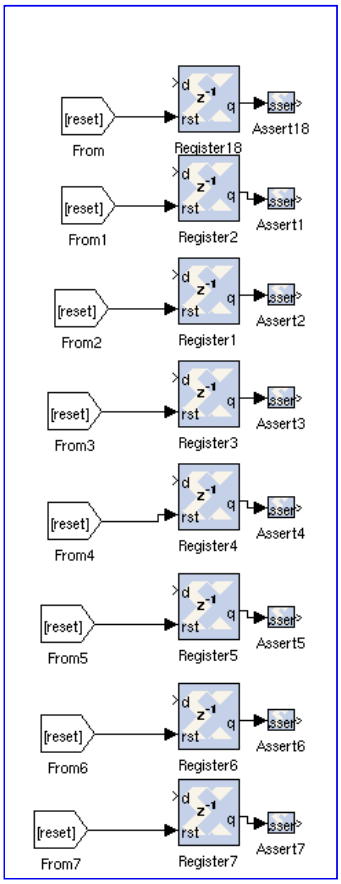
Duplicate the 'Assert' blocks for all the 18 registers

Click the block and drag it along with the 'Ctrl' key.

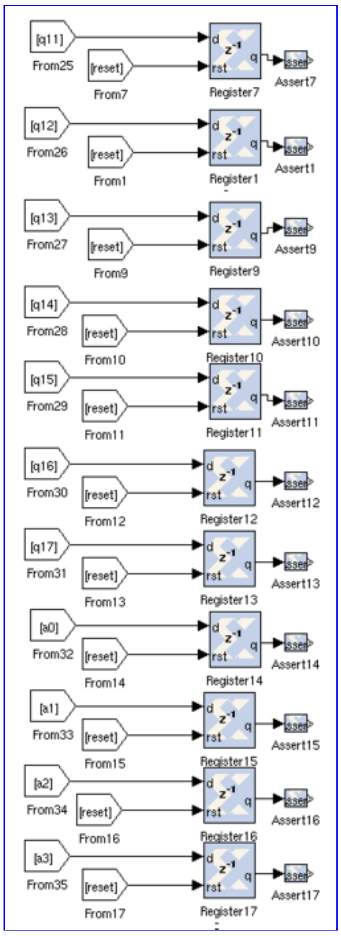


Add reset and input signals

Provide a common reset signal to all the registers

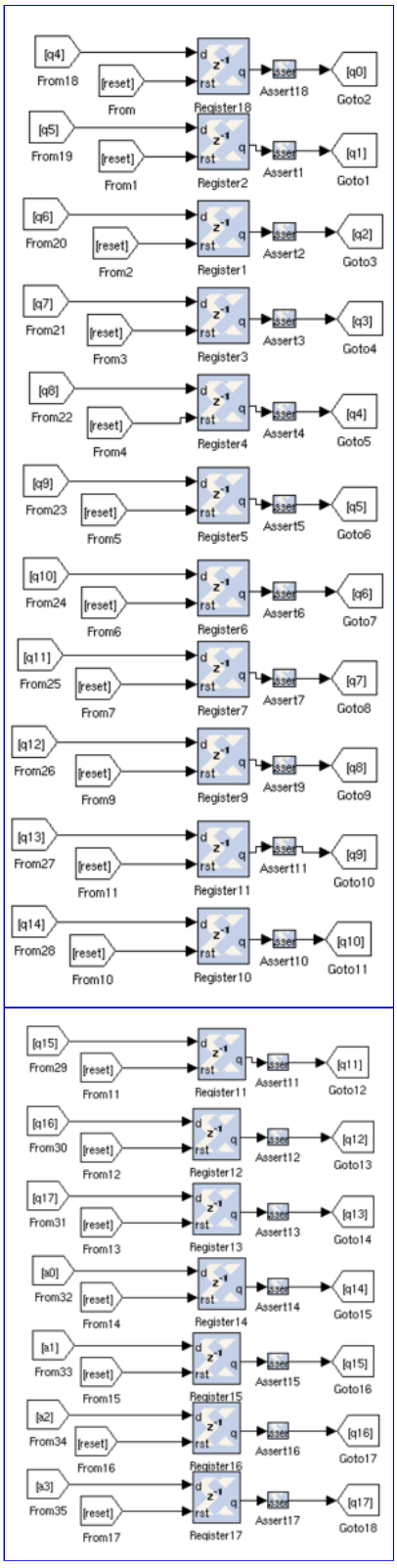


Provide inputs starting from q4 to q17 (above) and remaining from a0 to a3 (as shown below).



Add outputs

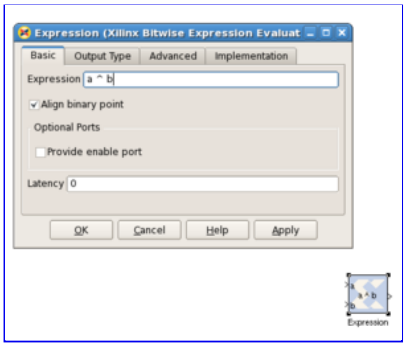
Connect the outputs of the register coming via the assert block starting from q0 to q17.



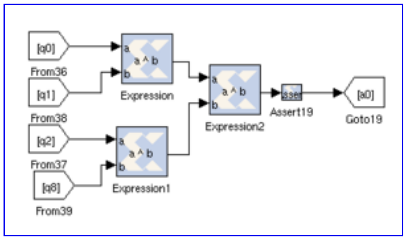
Leap Forward Logic

For leap forwarding to get parallel outputs from the LFSR, we need to add XORing logic. Add an Expression block from the Xilinx library and double click to change the expression to a ^ b. Keep the output type to

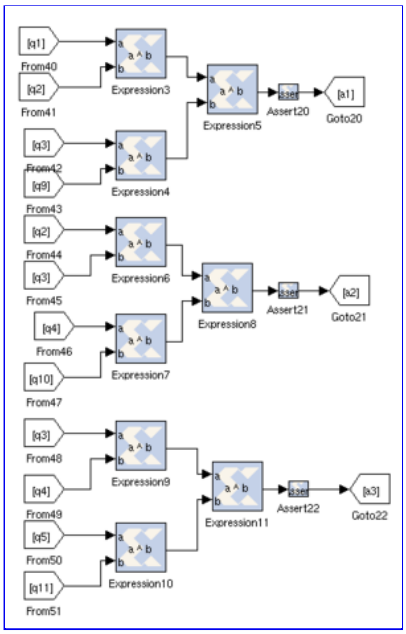
“Full”.



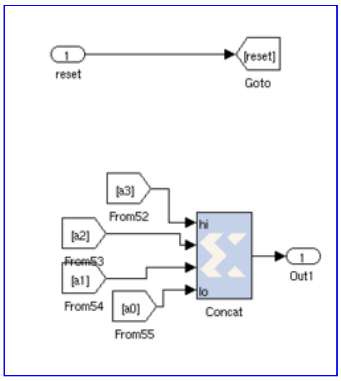
Using the expression blocks, create a structure for the LSB of the output as shown in the figure below.



Similarly, repeat it for all the other outputs as shown in figure below.

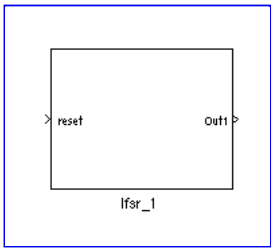


Concatenate the bits to get 4-bit output and also provide an input port for reset.

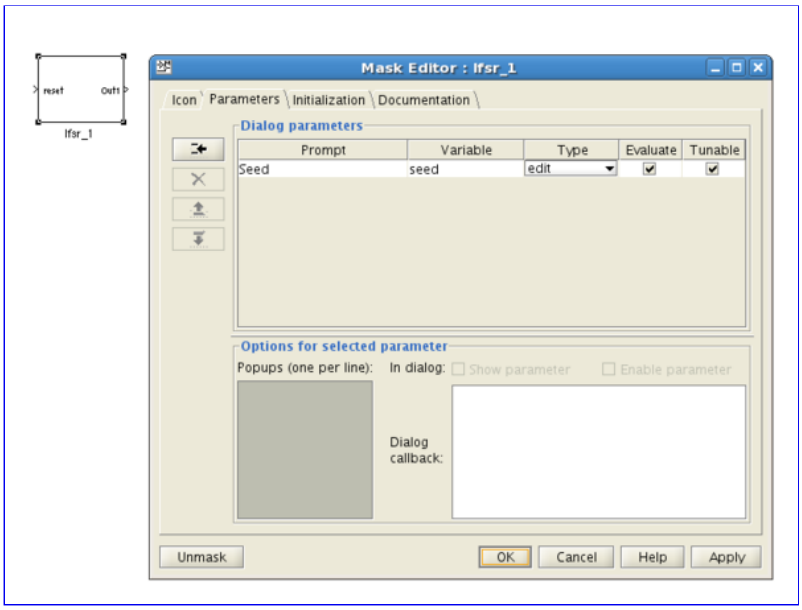


Creating subsystem for LFSR

Create a subsystem for this LFSR block (by selecting the entire logic and using right click to 'Create Subsystem'.

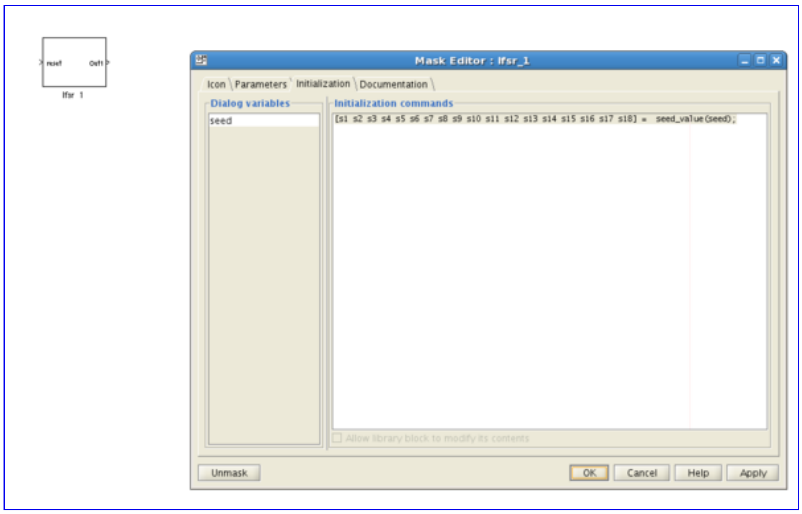


Once the subsystem is created, right click on the block and go to 'Edit Mask'. On clicking it, a window will open. Click on the first tab on the left column to add a new field in the table. Go to the 'Parameters' tab and add 'Seed' in the 'Prompt' field of the parameters. The variable would be set to 'seed' and Type to 'edit' and tick the 'Evaluate' and 'Tunable' field.



Now go to the initialization tab in the same window. In the field Dialog Variables, put 'seed' and in Initialization commands, paste the following line -

[s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18] = seed_value(seed);

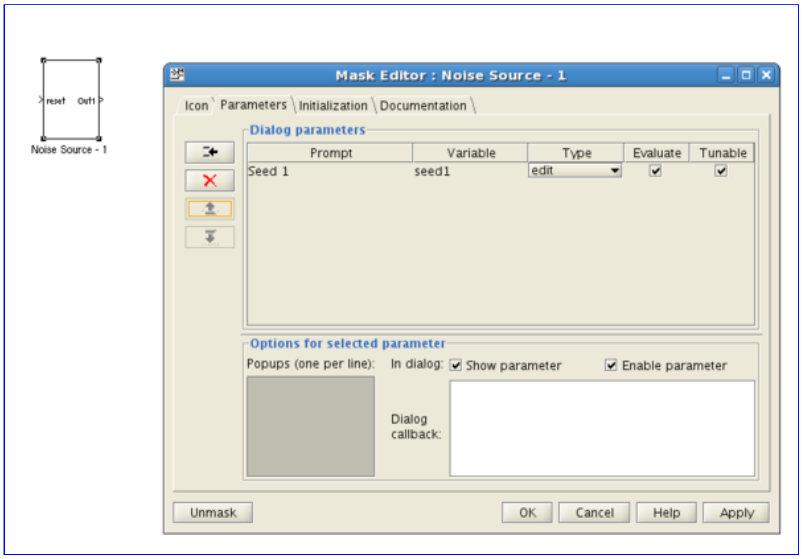


Steps mentioned above provide the seed value to the LFSR. This would be passed through a Matlab function, the file containing this Matlab function (seed_value.m).

We have one noise source ready. In order to check the probability distribution of this source, we will first have to pass a seed from the top level subsystem and also we will need to store the data into a memory in order to be read through a PC and analyzed.

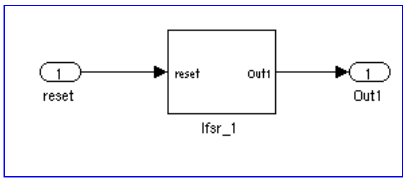
Preparing the LFSR block for compilation

To begin, create a subsystem that contains the LFSR block. As explained earlier, right click the subsystem and go to 'Edit Mask'. Add new field and add the details as shown in figure below –



The purpose of this tutorial is to develop the random number generator and also to understand the Central Limit Theorem, the concept behind generation of Gaussian distribution from uniform distribution. In order to view the distribution, we will need to add additional blocks so that the data from the random number generator can be stored in a memory and read to the P.C. using a script.

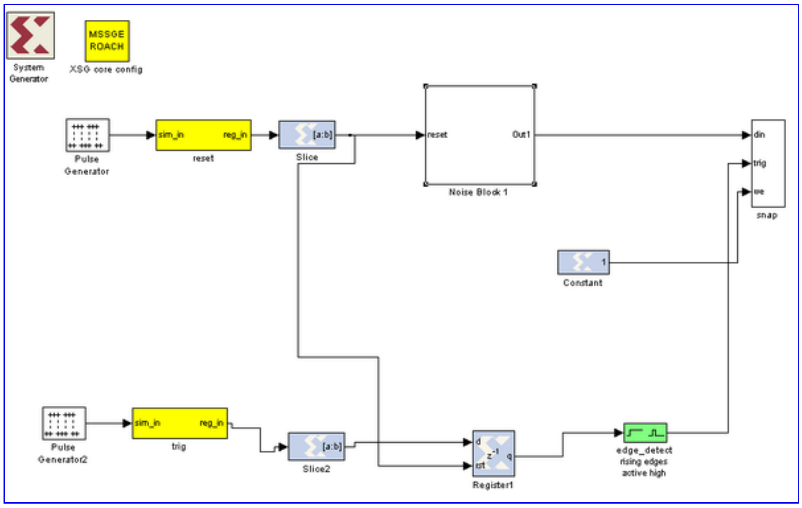
Converting the LFSR block into a subsystem



Repeat step 4.9 every time you add a new LFSR to the subsystem. Once the mask is edited add the value '33072' to the field. This is the random seed which will get converted to an 18-bit binary number and will be provided as an initial value to each of the 18 registers.

Adding software registers & snap block

Now, we need to provide reset to the circuit using software register, so add a software register, followed by a bit slice so that the LSB is provided to the reset. Please name the register 'reset'. You may add a Pulse Generator to the sim_in input of the software register. To store the generated random number samples, add a snap block from the CASPER library. This block will need trigger and write enable inputs. Provide write enable as a constant boolean input using a 'Constant' block. Trigger input can be provide similar to the reset but by adding a register (name it 'trig') followed by a rising-edge detector block. Also add the ROACH XSG Core Config and System Generator tokens.



Compiling the design

By giving bee_xps command in the matlab window we will get a pop-up. Make sure the file displayed in the pop-up is correct and then press RUN to start the compilation. After compilation , it creates a directory named after the model file name except .mdl and another sub directy as bit_files. In this bit_files directory there are .bit and .bof file. We need the .bof file to program the FPGA. You need to copy this .bof file at location [FPGA_PROG_BOF_DIR] after changing the permissions of the file.

eg. for the bof file [TUT6_BOF_FILE] in the area [STD_BOF_DIR]

$ chmod a+x [STD_BOF_DIR]/[TUT6_BOF_FILE]

$ cp [STD_BOF_DIR]/[TUT6_BOF_FILE] [FPGA_PROG_BOF_DIR]

Running the design

The python scripts are located in the “[STD_PYSCRIPT_DIR]” directory. We first need to run

“[TUT6_PYSCRIPT_FILE]” to program the FPGA and plot the result.

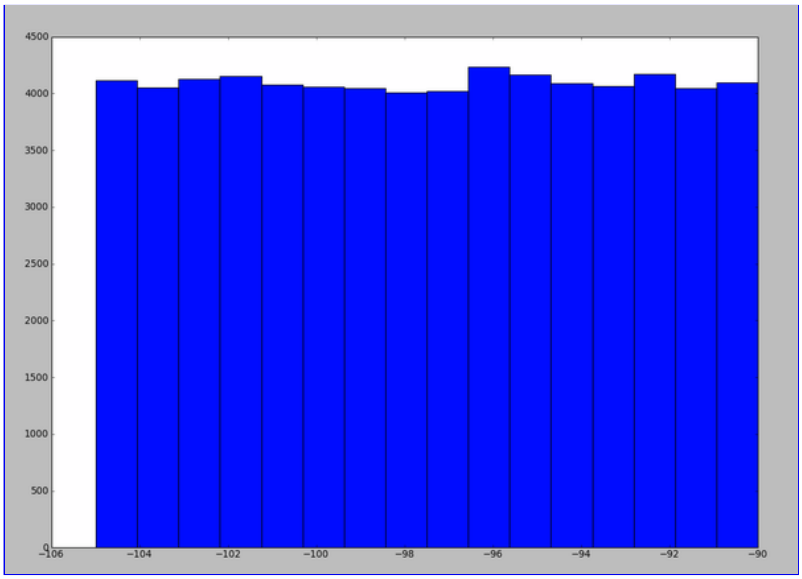
Usage : [STD_PYSCRIPT_DIR]/[TUT6_PYSCRIPT_FILE] <ROACH name/IP> -b <bof file>

eg.

$ [STD_PYSCRIPT_DIR]/[TUT6_PYSCRIPT_FILE] roach030172 -b [tut6_digital_noise_src_2011_Sep_23_1128.bof]

```
Enter the corresponding Location/File names and roach name/IP.
```

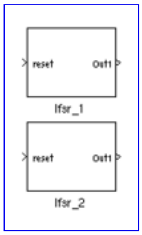
On the running above script, for a single noise source, a window showing the histogram of uniform distribution would appear as shown in the figure below -



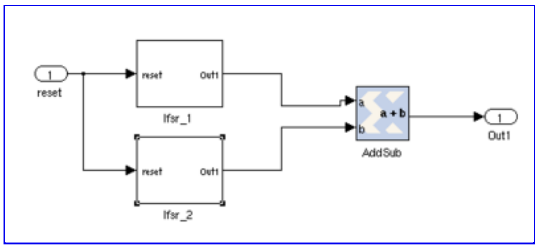
Adding more LFSR blocks to the design

We will need to add more LFSR blocks in the noise source and add them to make its distribution Gaussian. Now, changes are only to be made in the noise source block. Other subsystem blocks and main design would not need any change.

Within the noise source block, in order to get another LFSR block, use Ctrl + left click + drag the mouse. This will generate an identical copy of the block.



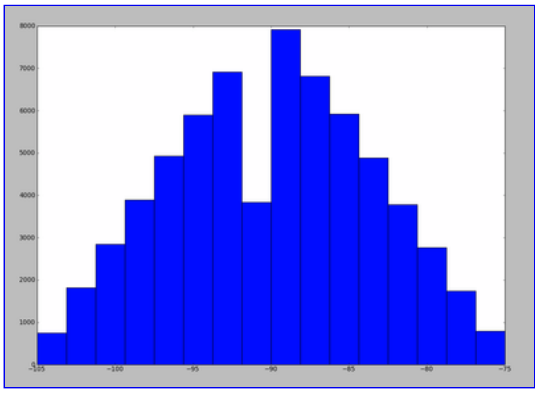
Now we need to check the output with two noise sources added. Get the adder block from the library and set the output type to 5-bit unsigned (since the two inputs are 4-bit unsigned).



Also double click the block and change the seed value to seed2. Repeat step 4.9. In step 4.10, add '2871' as the random seed.

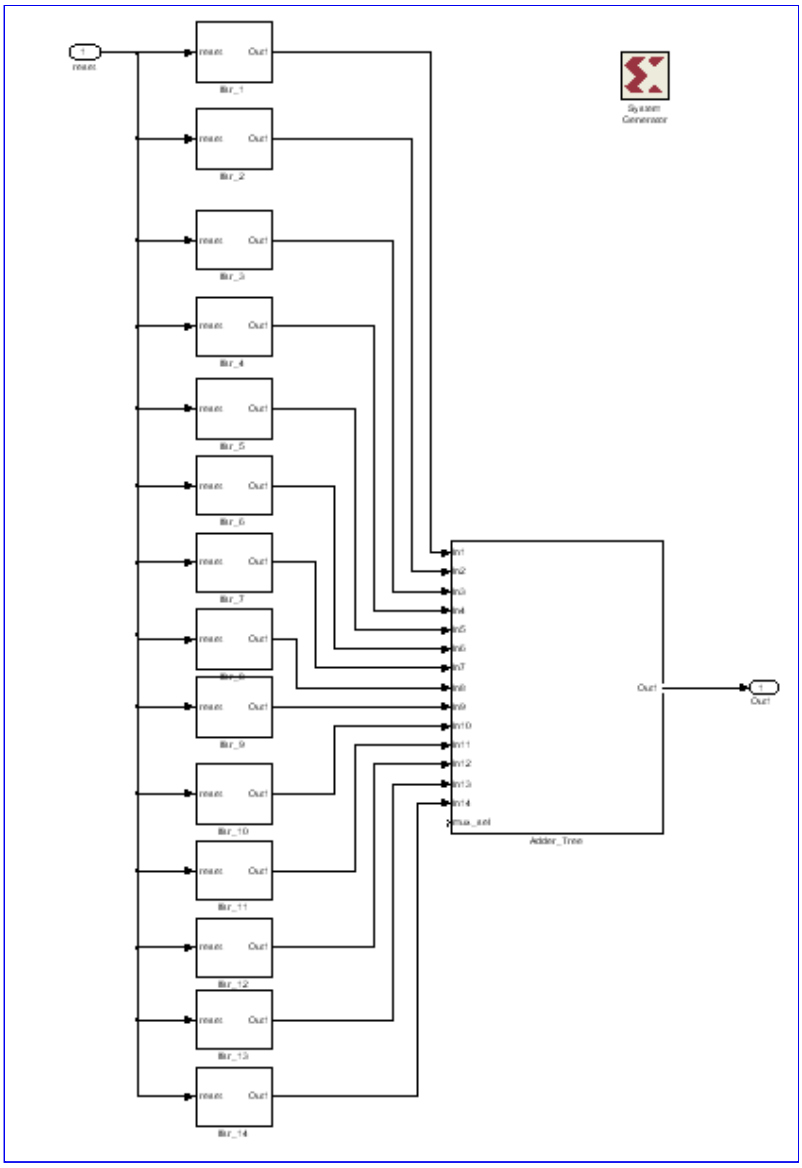
Repeat step 6 and check the histogram output.

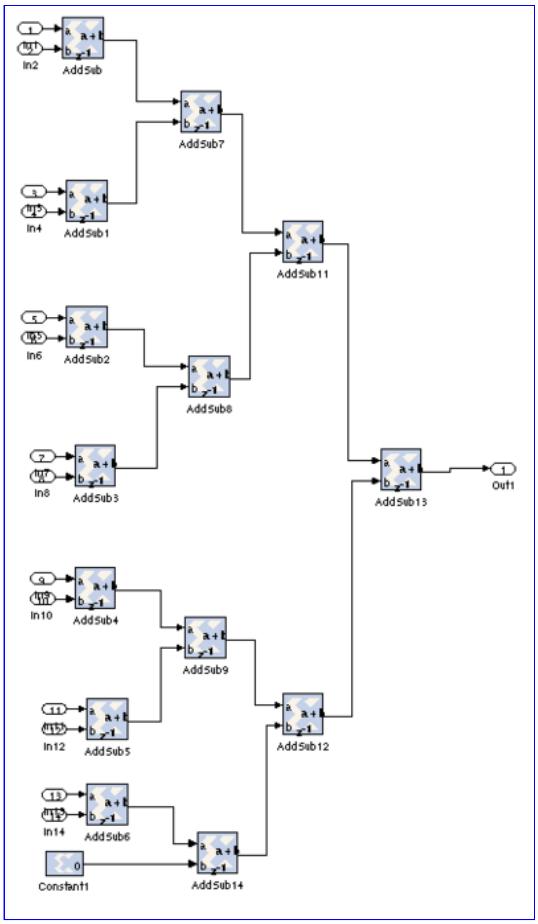
The output after adding two LFSRs (uniform number generators) should be as shown in the figure -



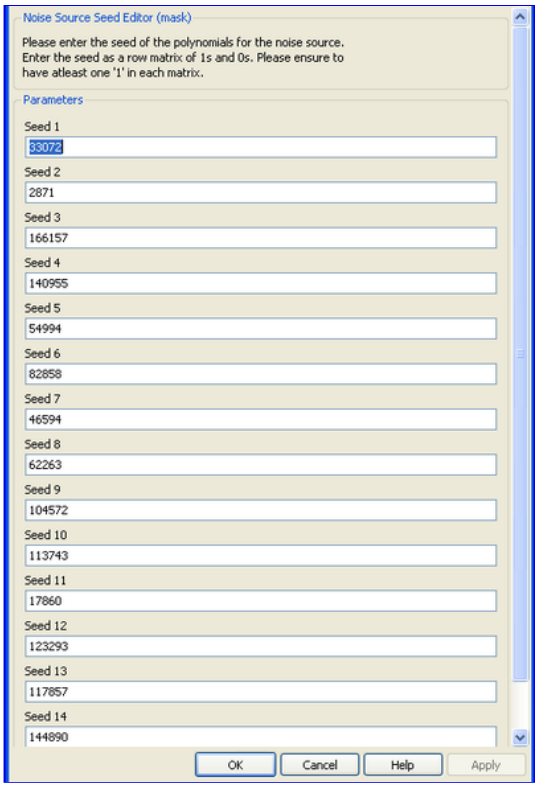
Towards Gaussian Output

Now in order to get Gaussian we will need to add 14 LFSRs. As mentioned earlier, instantiate the LFSRs and change the seed values. Now make an adder tree, each adder having two inputs and remember that the number of output at each stage would have one bit more than the number of input bits.

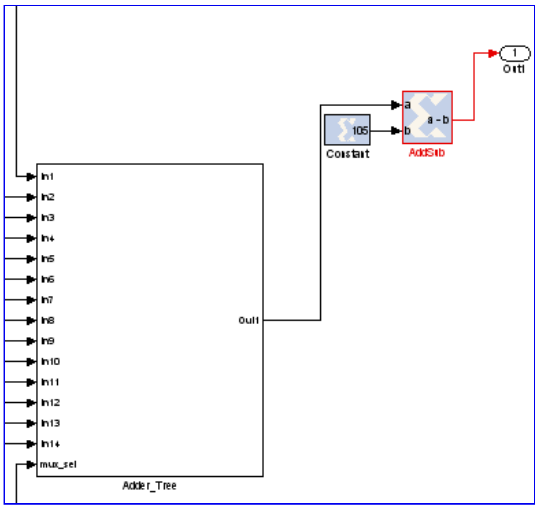

reset
Out1
Bit_1
Bit_2
Bit_3
Bit_4
Bit_5
Bit_6
Bit_7
Bit_8
Bit_9
Bit_10
Bit_11
Bit_12
Bit_13
Bit_14
System Generator
Adder_Tree
Out1



Repeat steps 4.9 and add the values of random numbers as shown in the snapshot below -

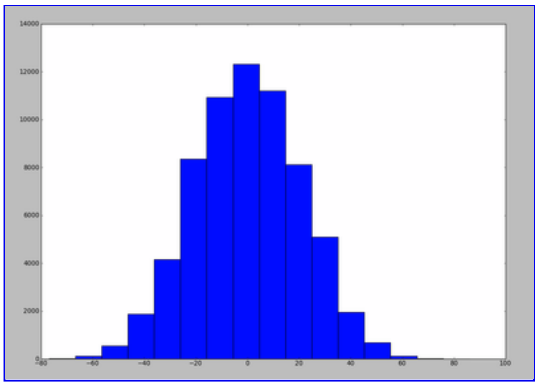


In order to make it zero mean subtract a constant 105 from the adder tree output as shown in figure below



Repeat step 6 and check the histogram output.

The histogram should appear as shown in the figure below -



Conclusion

This tutorial discussed the basics of digital noise source designed using combination of uniform random number generators. It was shown how to implement a digital noise source using the CASPER tool-flow and test it on ROACH board. The noise data was read from the FPGA to generate histogram to see the distribution of noise and also to understand the Central Limit Theorem.

Retrieved from "https://casper.berkeley.edu/wiki/Tutorial_-_Noise_Source"

Views

- Page
- Discussion
- View source
- History

Personal tools

- Log in

reference

- Main Page
- Libraries
- Toolflow
- Hardware
- Software

documents

- Projects
- Tutorials
- Memos
- Papers
- Videos
- FAQ

wiki

- Recent changes
- Random page
- Help

Search

Go Search

Toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link

- This page was last modified on 27 September 2011, at 14:50.
- This page has been accessed 272 times.
- Privacy policy
- About Casper
- Disclaimers