FPGA IMPLEMENTATION OF CORRELATION DEPENDENT STOCHASTIC CIRCUITS

Thesis is submitted in the partial fulfillment of the requirements for the degree of

MASTER IN ELECTRONICS AND TELECOMMUNICATIONS ENGINEERING

by

ARGHYAJOY MONDAL

Class Roll Number: 002010702008

Registration Number:136204 of 2016-2017

Examination Roll Number:M4ETC22008B

Under the Guidance of

Prof. MRINAL KANTI NASKAR

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

FACULTY COUNCIL OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

KOLKATA-700032

NOVEMBER, 2022

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

This is to certify that **Mr. Arghyajoy Mondal** (Class Roll No. **002010702008** and Registration No **136204 of 2016-2017**) bearing Examination Roll No.**M4ETC22008B** has completed and submitted his thesis entitled, "**FPGA IMPLEMENTATION OF CORRELATION DEPENDENT STOCHASTIC CIRCUITS**", in partial fulfilment of the requirements for the degree of "**Master in Control Engineering**" under Electronics and Telecommunications Engineering Department of Jadavpur University. The thesis work has been carried out by him under my guidance and supervision. The project, in my opinion, is worthy of its acceptance for the partial fulfilment of the requirement for the degree of Master of Electronics and Telecommunication engineering.

_____

SUPERVISOR
Prof. Mrinal Kanti Naskar
Professor
Department of Electronics and Telecommunication
Jadavpur University
Kolkata -700032

_____

Prof. Manotosh Biswas
Head of the Department
Electronics and Telecommunication
Engineering
Jadavpur University
Kolkata -700032

_____

Prof. Bhaskar Gupta
Dean
Faculty Council of
Engineering & Technology
Jadavpur University
Kolkata -700032

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL

The foregoing thesis, entitled "**FPGA IMPLEMENTATION OF CORRELATION DEPENDENT STOCHASTIC CIRCUITS**", is hereby approved as a creditable study of "**Master in Control Engineering**" under Electronics and Telecommunication Engineering department and presented by **Mr. ARGHYAJOY MONDAL** bearing Examination Roll No.**M4ETC22008B** in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion therein but approve this thesis only for the purpose for which it is submitted.

Committee on Final examination for evaluation of thesis:

Signature of Examiners

_____

Additional Examiner

_____

Supervisor

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original work by the undersigned candidate as part of his Master of Electronics and Telecommunication Engineering studies.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name **: ARGHYAJOY MONDAL**

Examination Roll No **: M4ETC22008B**

Class Roll No **: 002010702008**

Thesis Title : FPGA IMPLEMENTATION OF CORRELATION DEPENDENT STOCHASTIC CIRCUITS

Signature of the Candidate

Acknowledgement

Firstly, I would like to express my special gratitude to my project guide, Prof. Mrinal Kanti Naskar, for his supervision, invaluable suggestions, and continuous encouragement throughout the entire course of the research work. His valuable guidance and feedback has helped me in completing this project.

I am very grateful to Ms.Shyamali Mitra(Assistant Professor, Department of Instrumentation and Electronics Engineering)for her necessary suggestions and valuable help during my project work. I also wish to express my thanks to all my teachers who taught me during the tenure of my course work.

I would like to express my thanks to my senior Mr. SOUMYARUP PAL whose thesis work helped me to get a quick start in my project.

Above all, I extend my deepest gratitude to my parents for their invaluable love, affection, encouragement, and support and for their struggle to make me a successful person in spite of many obstacles. Their help continues to guide me in my life.

Lastly, I would like to thank Prof. Manotosh Biswas, head of the department, Electronics and Telecommunication Engineering Department, Jadavpur University, for his kind cooperation throughout the entire course.

ARGHYAJOY MONDAL

Abstract

Stochastic computing exploits random bit streams, realizing area-efficient hardware for complicated functions such as multiplication and tanh, as compared with more traditional binary approaches.

Unlike deterministic computing, stochastic computing does not assume that hardware always produces the same results if given the same inputs. It allows for noise and uncertainty (both in the inputs and in how the hardware operates), and tries to use them creatively.

Typically, each bit of an N-bit stochastic number (SN) X is randomly chosen to be 1 with some probability P(X), and X is generated and processed by conventional logic circuits. For instance, a single AND gate performs multiplication. The value X of an SN is measured by the density of 1s in it. SC has uses in massively parallel systems and is very tolerant of soft errors.

In this Thesis we have first briefly studied about SC, its generation which includes like random number generation, derandomizer unit, correlation among input values, etc and have subsequently developed stochastic computational
elements for certain operations.
Finally implemented some of the stochastic circuits in hardware either using circuitmaker and breadboard or FPGA.

Contents

CHAPTER 1 INTRODUCTION

Stochastic computing (SC) was proposed in the 1960s as a low-cost alternative to conventional binary computing. It is unique in that it represents and processes information in the form of digitized probabilities[1].

Modern computing hardware is constrained by stringent application requirements like extremely small size, low power consumption, and high reliability.

Stochastic computing (SC) is an unconventional method of computation that treats data as probabilities. Typically, each bit of an N-bit stochastic number (SN) X is randomly chosen to be 1 with some probability P(X), and X is generated and processed by conventional logic circuits. For instance, a single AND gate performs multiplication. The value X of an SN is measured by the density of 1s in it,

For example, a bit-stream a real number X containing 50% 1s and 50% 0s denotes the number P(X) = 0.5, reflecting the fact that the probability of observing a 1 at an arbitrary bit position is P(X). Neither the length nor the structure of X need be fixed.

In previous work, Gaines [2] introduced the concept of stochastic computing. Brown and Card [3] presented stochastic computational elements for neural computations,
Qian et al [4] proposed architecture or synthesis of stochastic circuits and polynomial combination using combinational logic. Najafi et al [5] showed high speed stochastic circuits using synchronous analog pulses.

The thesis is organized as follows. In chapter 2 we discuss some general concepts of Stochastic circuits of SC. We will discuss briefly about stochastic process and as well the accuracy of stochastic representation. Chapter 3 deals with generation of pseudorandom noise, SN formats, combinational circuits using stochastic logic.

Chapter 4 deals with the Correlation dependence of stochastic numbers and circuits. In chapter 5 we have listed down all the electronic design tools and hardware that we have using during the implementation of the stochastic circuits. Chapter 6 consists of Hardware implementation of stochastic components that has been used in the SC.

In chapter 7 hardware implementation of stochastic combinational circuits takes place. Chapter 8 deals with the Stochastic greatest common divisor , its hardware implementation along with FPGA implementation. In chapter 9 we have implemented a stochastic number generator with a specified stochastic correlation coefficient. Finally chapter 10 deals with the conclusion and the future scope with the proposed research and chapter 11 consists references.

CHAPTER 2:General Idea of Stochastic Computing

In this chapter we will get a general overview of the stochastic computing, numbers and circuits.

2.1 Stochastic Computing and processes

Stochastic computing is a collection of techniques that represent continuous values by streams of random bits. Complex computations can then be computed by simple bit-wise operations on the streams.

In simple words Statistical processes like tossing of coin, movement of gas molecules where outcomes does not follow any strict pattern instead they are unpredictable are referred to as stochastic process.

2.2 Advantages and Disadvantages of stochastic circuits

There are many advantages in using stochastic circuits over conventional binary bits operated circuits is that is that it enables very low-cost implementations of arithmetic operations using standard logic elements. For example, multiplication can be performed by a stochastic circuit consisting of a single AND gate.

Another attractive feature of SC is a high degree of error tolerance, especially for transient or soft errors caused by process variations or cosmic radiation. A single bitflip in a long bit-stream will result in a small change in the value of the stochastic number represented. For example, a bit-flip in the output of the changes its value from 3/8 to 4/8 or 2/8, which are the representable numbers closest to the correct result. But if we consider the same number 3/8 in conventional binary format 0.011, a single bit-flip causes a huge error if it affects a high-order bit. A change from 0.011 to 0.111, for example, changes the result from 3/8 to 7/8. [1]Stochastic numbers have no high-order bits as such since all bits of a stochastic bit-stream have the same weight.

On the other hand, SC has several problems that have severely limited its practical

applicability to date. An increase in the precision of a stochastic computation requires an exponential increase in bit-stream length, implying a corresponding exponential increase in computation time. For instance, to change the numerical precision of a stochastic computation from 4 to 8 bits requires increasing bit-stream length from 2^4 = 16 bits to 2^8 = 256 bits. variations in the representation of
the numbers being processed can lead to inaccurate results. An extreme case occurs when identical bit-streams denoting some number p are applied to the AND gate: the
result then is P, rather than the numerically correct product p × p = p^2.

CHAPTER 3 Stochastic Numbers and Circuits

The stochastic numbers are represented by bit streams containing 0 and 1 and interpreted through probability. The representation of bit-streams is random and independent, but can be dependent on the requirements of arithmetic operations.

Since stochastic numbers are treated as probabilities, they fall naturally into the interval [0,1].

2.3 Coding formats used in stochastic computing

There are mainly 3 coding formats used in stochastic computing which are

Unipolar coding method
Bipolar coding method
The ratio's of 1's and 0's

3.1.1 Unipolar coding method

The stochastic representation is based on the fraction of 1's in bit streams. Stochastic logic was proposed by Gaines in two formats: unipolar and bipolar [2].

In the unipolar format, a real number x is represented by a stochastic bit stream X,

Where

$x = p(X = 1) = p(X)$. Since x corresponds to a probability value, the unipolar representation must satisfy $0 \leq x \leq 1$.

3.1.2 Bipolar coding method

In the bipolar format, a real number x is represented by a stochastic bit stream X,

Where,

$x = 2p(X = 1) - 1 = 2p(X) - 1$, where $-1 \leq x \leq 1$.

3.1.3 Ratio of 1's and 0's

This concept was first proposed by Lee et al [6]. The format uses the ratio of number of 1's and 0's to represent any number in the range $[0,\infty)$.
Any positive number say, X can be represented by

$P(X) = N1 / N0$ N1 = Number of 1's in the bit stream X, N0 = Number of 0's in the stream.

The corresponding value in the unipolar domain for this representation will be $P(X) /(1-P(X))$.

3.2 Fundamental elements used in stochastic circuits

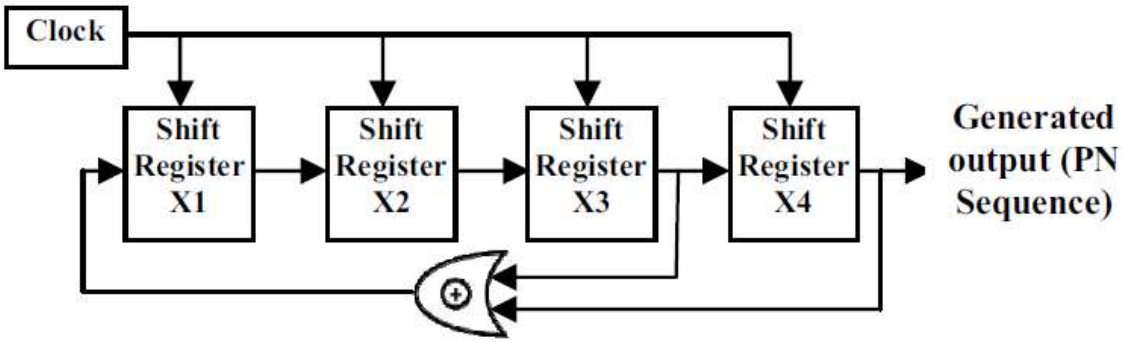
Basic elements or blocks of a stochastic circuits must contain a pseudorandom number generator which is being used in the design of a stochastic number generator , combinational or logic gates and a de randomizer which converts the operational stochastic bit stream into the binary number again.

To reduce such inaccuracies, SNGs are needed which produce stochastic numbers that are sufficiently random and uncorrelated. These requirements can be met by linear feedback shift registers (LFSRs).

3.2.1 Linear Feedback Shift Register

LFSR is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is XOR. Thus, an LFSR is most often a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value.[7]

The initial value of the LFSR is called the seed. Because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle. Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. [8]



The bits in the LFSR state which influence the input are called taps. A maximum-length LFSR produces an m sequence (i.e. it cycles through all possible 2n -1 states within the shift register except the state where all bits are zero), unless it contains all zeros, in which case it will never change. The sequence of numbers generated by this method is random.

The period of the sequence is (2n - 1), where n is the number of shift registers used in the design.

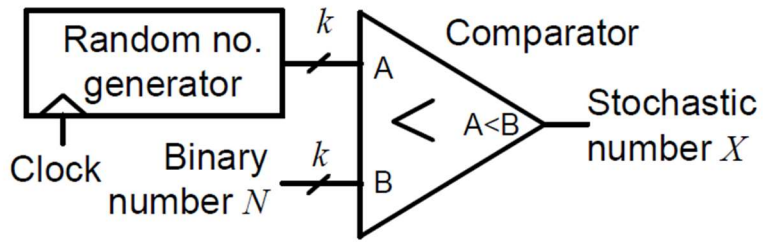
The below table depicts possible and maximum length polynomial:

Size of LFSR	Possible Feedback Polynomial	Maximum Length Feedback polynomial
8 Bit	$X^8 + X^7 + 1$, $X^8 + X^5 + 1$, $X^8 + X^7 + X^6 + X^5 + 1$, $X^8 + X^6 + X^4 + X^3 + X^2 + X^1 + 1$, etc	$X^{16} + X^{14} + X^{13} + X^{11} + 1$
16 Bit	$X^{16} + X^{15} + 1$, $X^{16} + X^{13} + X^{12} + X^9 + 1$, $X^{16} + X^{11} + X^{10} + X^7 + X^3 + X^1 + 1$, $X^{16} + X^{15} + X^{14} + X^{12} + X^7 + X^6 + + X^3 + X^2 + 1$, etc	$X^{16} + X^{14} + X^{13} + X^{11} + 1$
32 Bit	$X^{32} + X^{31} + 1$, $X^{32} + X^{28} + X^{27} + X^9 + 1$, $X^{32} + X^{21} + X^{15} + X^{13} + X^{12} + X^{10} + + X^8 + X^4 + 1$, $X^{32} + X^{31} + X^{27} + X^{24} + X^{19} + X^{18} + + X^{17} + X^{14} + X^{13} + X^{11} + X^5 + X^4 + X^1$, etc	$X^{32} + X^{22} + X^2 + X^1 + 1$

3.2.2 Stochastic number generator

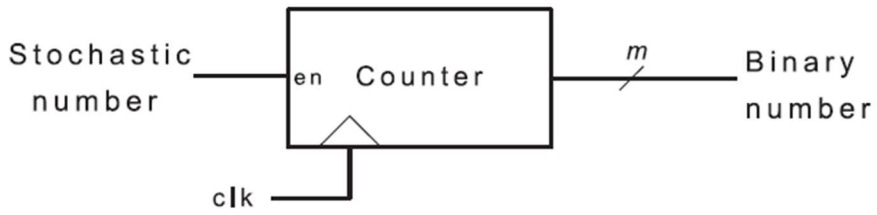
SNG or stochastic number generator is use for generating stochastic number by using a pseudorandom generator and a comparator(Although there are many other techniques available to generate stochastic number [1].)

The conversion process involves generating an *m*-bit random binary number in each clock cycle by means of a random or, more likely, a pseudorandom number generator, and comparing it to the *m*-bit input binary number. The comparator produces a 1 if the random number is less than the binary number and a 0 otherwise.



3.2.3 De-Randomizer Unit

After the conversion of the binary number into the stochastic number , we need to convert the stochastic number again back to binary fort this task we need help of De-randomizer Unit which is nothing but to count the number of 1's available in the bi stream.
For that task we just need a counter to count the number of 1's in the bit stream.



CHAPTER 4: Correlation dependence of stochastic numbers and circuits

In this chapter we will going to discuss the correlation dependency of the stochastic number and the circuits.

4.1 Correlation dependence of stochastic numbers

Correlation refers to statistical similarity between two phenomena. The correlation of two sequences (bit-streams) is measured by some form of covariance or dot-product operation.

With the measurement as follows

i)a correlation value of +1 means maximum similarity,
ii)a correlation value -1 means minimum similarity (maximum difference), and

iii)a correlation of 0 means the sequences are uncorrelated.

Now suppose *X* and *Y* are two bit streams of length *n, a* refers to the number of overlapping of 1s, *b* refers to the number of overlapping of 1s in *X* and 0s in *Y*, *c* refers to the number of overlapping of 0s in *X* and 1s in *Y* and *d* refers to the number of overlapping of 0s.
Then the Pearson correlation coefficient can be measured using the following formula:

$$\boldsymbol{\rho(X,\ Y) = \frac{ad - bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}}$$

But, this fails to capture the stochastic nature of bit streams for which Alaghi proposed SCC (Stochastic coefficient correlation).

Now for two stochastic numbers ***X*** and ***Y***, Stochastic Coefficient Correlation ***SCC(X, Y)*** is given by[13]

$$SC\ (X,\ Y) = \begin{cases} \dfrac{p_{x\wedge y} - p_x p_y}{\min(p_x,\ p_y) - p_x p_y} & if\ p_{x\wedge y} > p_x p_y \\ \dfrac{p_{x\wedge y} - p_x p_y}{p_x p_y\ - \max(p_x + p_y\ - 1, 0)} & otherwise \end{cases}$$

Where $x \wedge y$ is bit-wise AND function, and $p_{x\wedge y}$ is probability of 1's.

When stochastic numbers ***X*** and ***Y*** are uncorrelated to each other, $p_{x\wedge y} - p_x p_y = 0$, that means, $SSC\ (X,\ Y) = 0$, when ***X*** and ***Y*** are similar to each other (or maximum similarity), $SSC\ (X,\ Y) = +1$, and when ***X*** and ***Y*** are different to each other (or maximum difference), $SSC\ (X,\ Y) = -1$. The following table shows some examples of bit streams with their SCC values. In this thesis,

correlation is referred to as **r.**

Stochastic numbers	**SC correlation** $SCC\ (X,\ Y)$
X = 1111111100000000 Y = 1111000011110000	0
X = 1111111100000000 Y = 1111111100000000	+1
X = 1111111100000000 Y = 0000000011111111	-1

4.2 Correlation dependence of stochastic circuits

AND gate implements multiplication function, function of AND gate is, $\mathbf{p_z = F(p_x, p_y) = p_x * p_y}$, when **SCC(X, Y) = 0**. But, in presence of correlations, circuit behaviours changes, when **SCC(X, Y) = +1**, $\mathbf{p_z = F(p_x, p_y) = min(p_x, p_y)}$, when **SCC(X, Y) = -1,** $\mathbf{p_z = F(p_x, p_y) = max(p_x + p_y - 1, 0).}$

The following table shows the functions of some basic circuits with various correlation levels.

Circuits	**SCC = 0**	**SCC = +1**	**SCC = –1**
AND	$p_x \times p_y$	$\min(p_x, p_y)$,	$\max(p_x + p_y - 1, 0)$.
XOR	$(p_x + p_y - 2 \times p_x \times p_y)$	$\lvert p_x - p_y \rvert$	$\min(2 - p_x - p_y, p_x + p_y)$
OR	$(p_x + p_y - p_x \times p_y)$	$\max(p_x, p_y)$	$\min(p_x + p_y, 1)$
Multiplexer	$\frac{1}{2}(p_x + p_y)$	$\frac{1}{2}(p_x + p_y)$	$\frac{1}{2}(p_x + p_y)$

This table shows that AND works as a multiplier when correlation is +1, XOR gate works as absolute subtractor when correlation is +1, Multiplexer works as a scaled adder irrespective of correlation. In the later chapter we will be going to exploit thid fact in the implementation of the stochastic GCD.

CHAPTER 5 ELECTRONICS DESIGN TOOL

In this chapter we discuss the design tools and software's we have use to implement various stochastic circuits either in physical manner or in Software.

i)CIRCUIT MAKER 2000:

So the first software we have used is the Circuit Maker 2000 pro,
Which is a sophisticated software for designing and simulating various analog as well as digital circuits.

ii)VIVADO 2020.2:

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE).

Components in VIVADO :

The Vivado High-Level Synthesis compiler
enables C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to manually create RTL. Vivado HLS is widely reviewed to increase developer productivity, and is confirmed to support C++ classes, templates, functions and operator overloading.

Vivado 2014.1 introduced support for automatically converting OpenCL kernels to IP for Xilinx devices. OpenCL kernels are programs that execute across various CPU, GPU and FPGA platforms.

The Vivado Simulator is a component of the Vivado Design Suite. It is a compiled-language simulator that supports mixed-language, Tcl scripts, encrypted IP and enhanced verification.

The Vivado IP Integrator allows engineers to quickly integrate and configure IP from the large Xilinx IP library.

The Integrator is also tuned for MathWorks Simulink designs built with Xilinx's System Generator and Vivado High-Level Synthesis.

The Vivado Tcl Store is a scripting system for developing add-ons to Vivado, and can be used to add and modify Vivado's capabilities. Tcl is the scripting language on which Vivado itself is based. All of Vivado's underlying functions can be invoked and controlled via Tcl scripts.

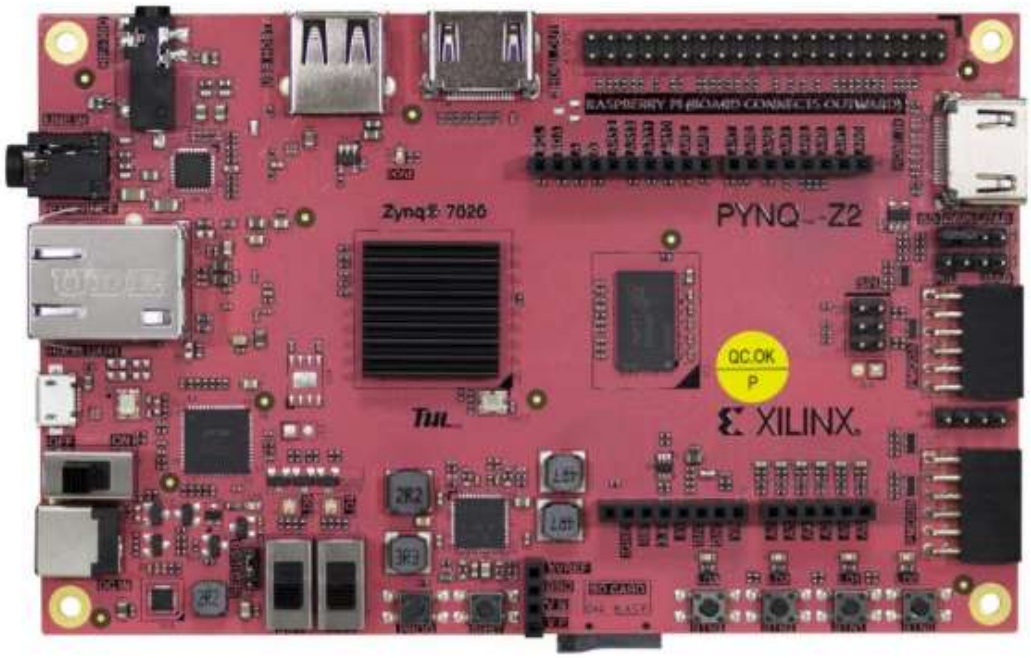
5.3 Hardware implementation tools

For physical implementation we used PYNQ-Z2 FPGA which is an open-source project from Xilinx that makes it easier to use Xilinx platforms. Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors to build more capable and exciting electronic systems.

- The PYNQ-Z2 board featuring the ZYNQ XC7Z020-1CLG400C SoC
- And consists of
- ZYNQ XC7Z020-1CLG400C o 650MHz ARM® Cortex-A9 dual-core processor o Programmable logic
- 13,300 logic slices, each with four 6-input LUTs and 8 flipflops ▪ 630 KB block RAM ▪ 220 DSP slices
- On-chip Xilinx analog-to-digital converter (XADC) o Programmable from JTAG, Quad-SPI flash, and MicroSD card
- Memory and storage o 512MB DDR3 with 16-bit bus @ 1050Mbps o 16MB Quad-SPI Flash with factory programmed 48-bit globally unique EUI-48/64™ compatible identifier o MicroSD slot
- Power o USB or 7V-15V external power regulator

- USB and Ethernet o Gigabit Ethernet PHY o Micro USB-JTAG Programming circuitry o Micro USB-UART bridge o USB 2.0 OTG PHY (supports host only)
- Audio and Video o 2x HDMI ports (input and output) o 24bit I2S DAC with 3.5mm TRRS jack o Line-in with 3.5mm jack
- Switches, Push-buttons and LEDs o 4 push-buttons o 2 slide switches o 4 LEDs o 2 RGB LEDs •
- Expansion Connectors o 2xPmod ports
- 16 Total FPGA I/O (8 pins on Pmod A are shared with Raspberry Pi connector) o Arduino Shield compatible connector
- 24 Total FPGA I/O
- 6 Single-ended 0-3.3V Analog inputs to XADC o Raspberry Pi connector
- 28 Total FPGA I/O (8 pins are shared with Pmod A)
- Other than FPGA we have used breadboards IC's and connecting wires for the hardware implementation.

- This picture below is PYNQ Z2 FPGA board



- Other than FPGA we have used breadboards IC's and connecting wires for the hardware implementation.

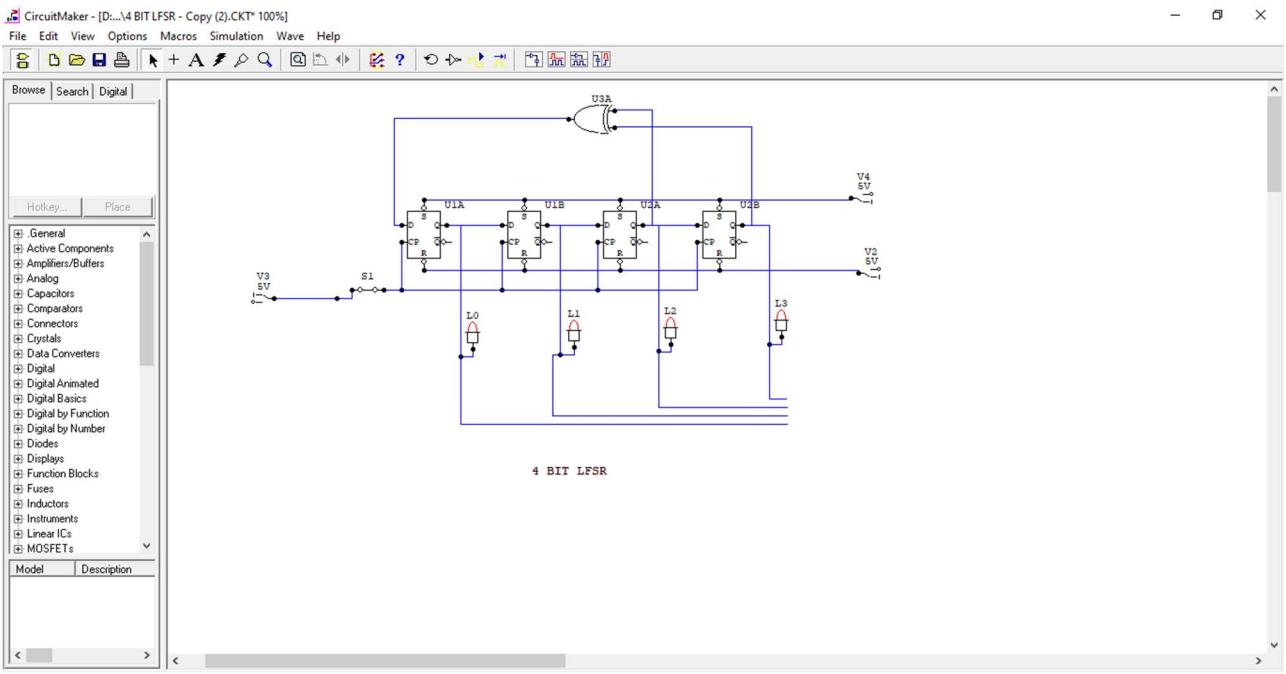
CHAPTER 6 Hardware implementation of stochastic components

In this chapter consists of the hardware implementation of stochastic circuits fundamental elements in both bread board and in Vivado software.

6.1 Linear Feedback Shift Register

6.1.1 LFSR Circuit maker Implementation

Here in this thesis we have implement a 4 bit LFSR using circuit maker 2000 and simulate the same.



The during the simulation we have got the following results:

L4	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1
L3	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1
L2	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1	1
L1	1	0	0	0	1	0	0	1	1	0	1	0	1	0	1	1

6.1.2 LFSR FPGA Implementation

To simulate in Vivado we choose VHDL as the HDL.
The Source code for the N bit LFSR is as follows

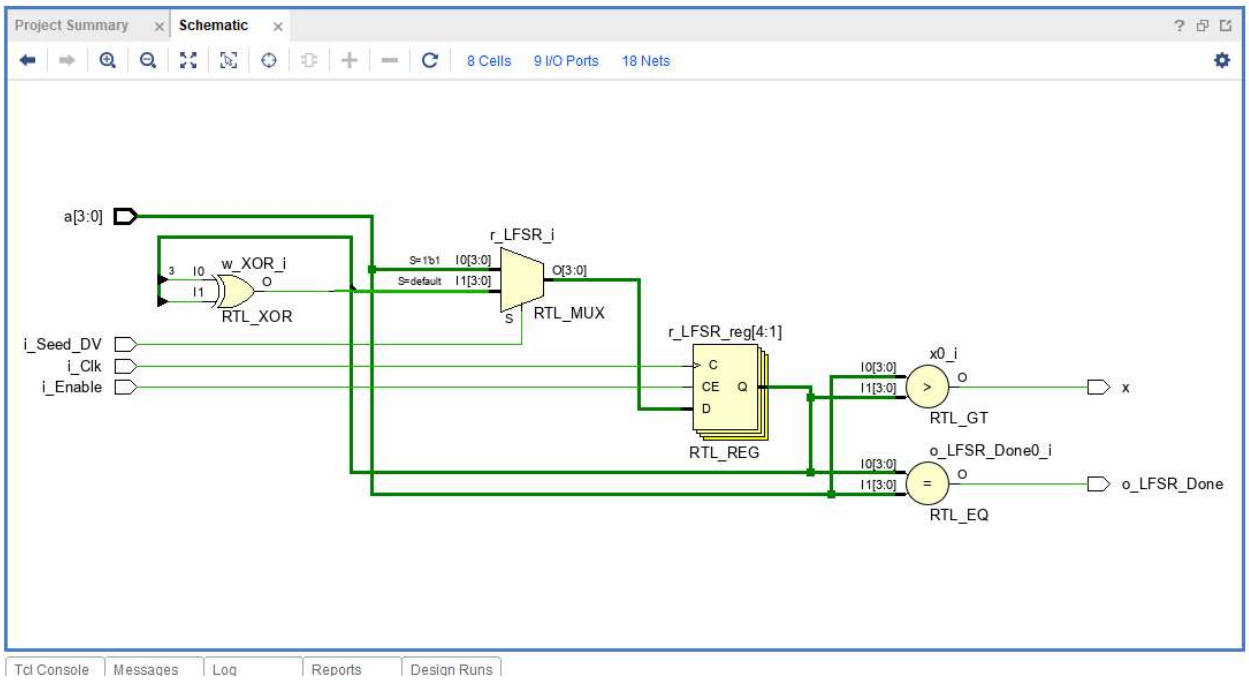
```
-- Company: JADAVPUR UNIVERSITY
-- Engineer: ARGHYAJOY MONDAL
--
-- Create Date: 07/13/2022 03:57:05 PM
-- Design Name:
-- Module Name: n_BIT_LFSR - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
```

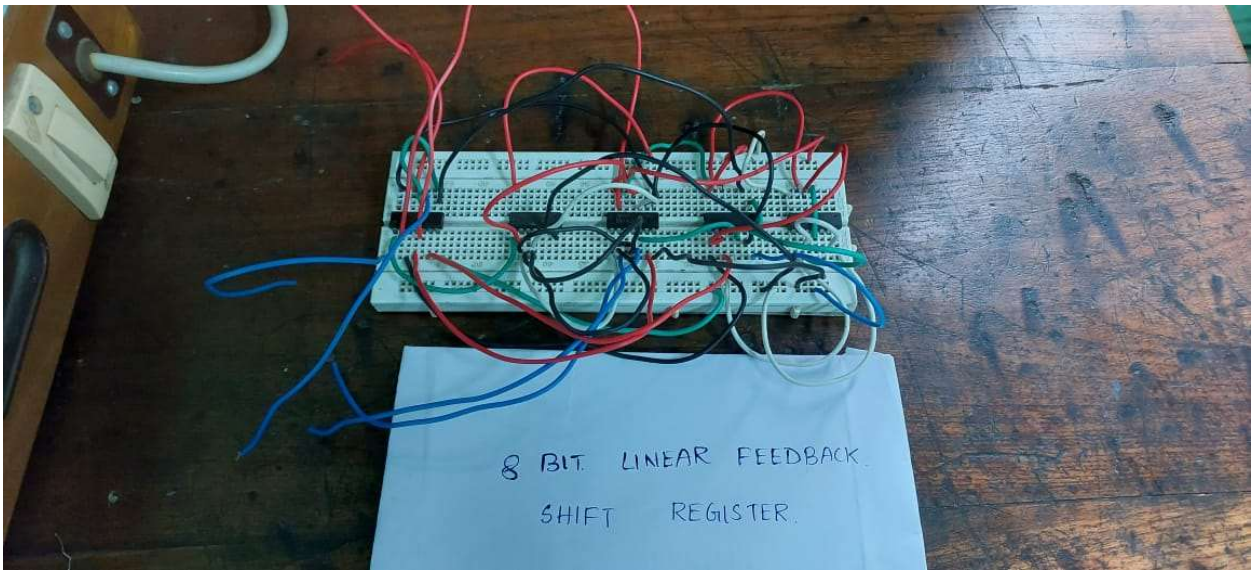
```
library ieee;
use ieee.std_logic_1164.all;

entity LFSR is
  generic (
    g_Num_Bits : integer := 4
    );
  port (
    i_Clk    : in std_logic;
    i_Enable : in std_logic;

    i_Seed_DV   : in std_logic;
    a : in std_logic_vector(g_Num_Bits-1 downto 0);


    x : out std_logic;

    o_LFSR_Done : out std_logic
    );
end entity LFSR;

architecture RTL of LFSR is

  signal r_LFSR : std_logic_vector(g_Num_Bits downto 1) := (others => '0');
  signal w_XOR : std_logic;
   signal b : std_logic_vector(g_Num_Bits-1 downto 0);
   signal i_seed_data :  std_logic_vector(g_Num_Bits-1 downto 0);



begin

  -- Purpose: Load up LFSR with Seed if Data Valid (DV) pulse is detected.
  -- Othewise just run LFSR when enabled.
  p_LFSR : process (i_Clk) is
  begin
    if rising_edge(i_Clk) then
      if i_Enable = '1' then
```

```
      if i_Seed_DV = '1' then
        r_LFSR <= a;
      else
        r_LFSR <= r_LFSR(r_LFSR'left-1 downto 1) & w_XOR;
      end if;
    end if;
  end if;
end process p_LFSR;

g_LFSR_3 : if g_Num_Bits = 3 generate
  w_XOR <= r_LFSR(3) XOR r_LFSR(2);
end generate g_LFSR_3;

g_LFSR_4 : if g_Num_Bits = 4 generate
  w_XOR <= r_LFSR(4) XOR r_LFSR(3);
end generate g_LFSR_4;

g_LFSR_5 : if g_Num_Bits = 5 generate
  w_XOR <= r_LFSR(5) XOR r_LFSR(3);
end generate g_LFSR_5;

g_LFSR_6 : if g_Num_Bits = 6 generate
  w_XOR <= r_LFSR(6) XOR r_LFSR(5);
end generate g_LFSR_6;

g_LFSR_7 : if g_Num_Bits = 7 generate
  w_XOR <= r_LFSR(7) XOR r_LFSR(6);
end generate g_LFSR_7;

g_LFSR_8 : if g_Num_Bits = 8 generate
  w_XOR <= r_LFSR(8) XOR r_LFSR(6) XOR r_LFSR(5) XOR r_LFSR(4);
end generate g_LFSR_8;

  b <= r_LFSR(r_LFSR'left downto 1);
o_LFSR_Done <= '1' when r_LFSR(r_LFSR'left downto 1) = a else '0';
 i_seed_data <= a;

process(a,b)
 begin
  if ( a > b) then
```

```
    x <= '1';
    else
    x <= '0';
    end if;
    end process;


end architecture RTL;
```

The RTL schematic which consists of LFSR is as follows:





The above picture is 8-bit LFSR implemented on a bread board

The Hardware implementation consists of these IC's

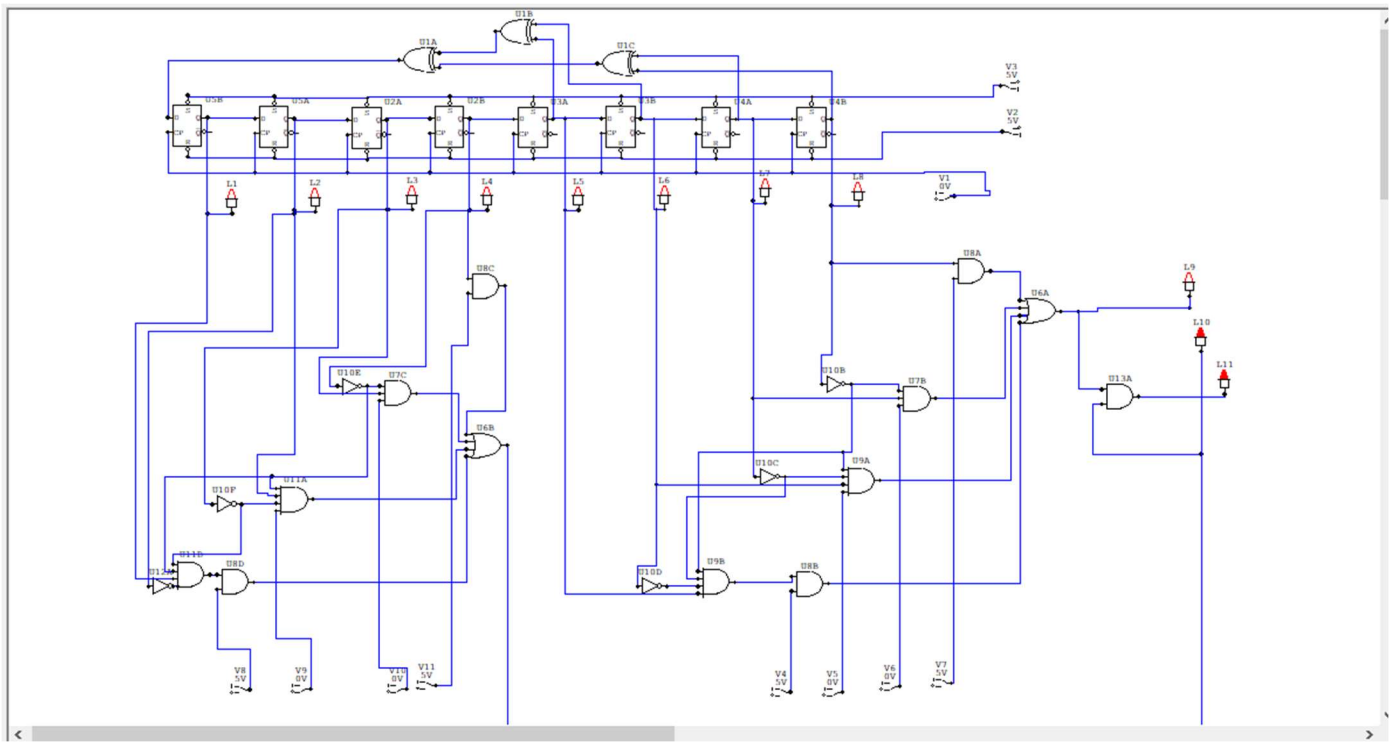
7486 ,7474 ,7404 , bread board, digital simulator and connecting wires.

6.2 Stochastic number generator

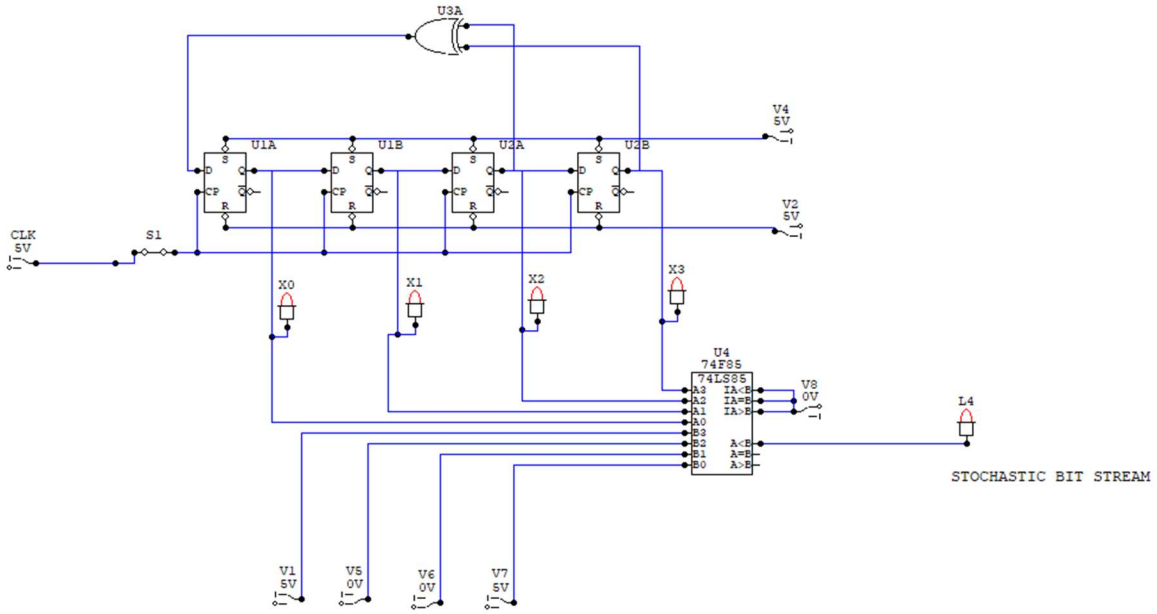
6.2.1 Stochastic number generator Circuit maker Implementation

Here In this thesis we have implemented a 8 bit SNG and a 4 bit SNG in circuit maker 2000

The circuit maker 2000 implementation of 8 bit SNG using the Weighted Binary Method is as follows:



The circuit maker 2000 implementation of 4bit SNG using the Comparator method is as follows:

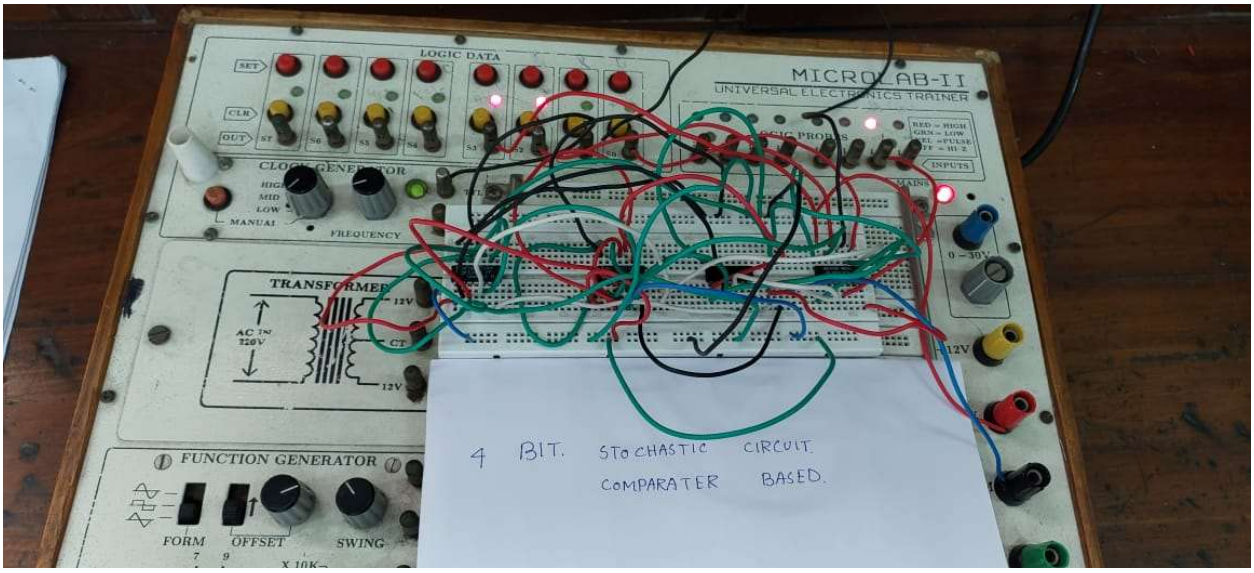


The results found by simulating the above circuit is in the below table: For the Bit stream of 1100

L4	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1
L3	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1
L2	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1	1
L1	1	0	0	0	1	0	0	1	1	0	1	0	1	0	1	1
X	1	1	1	1	1	1	0	1	1	0	1	1	0	1	0	1

Here X is the generated stochastic number.

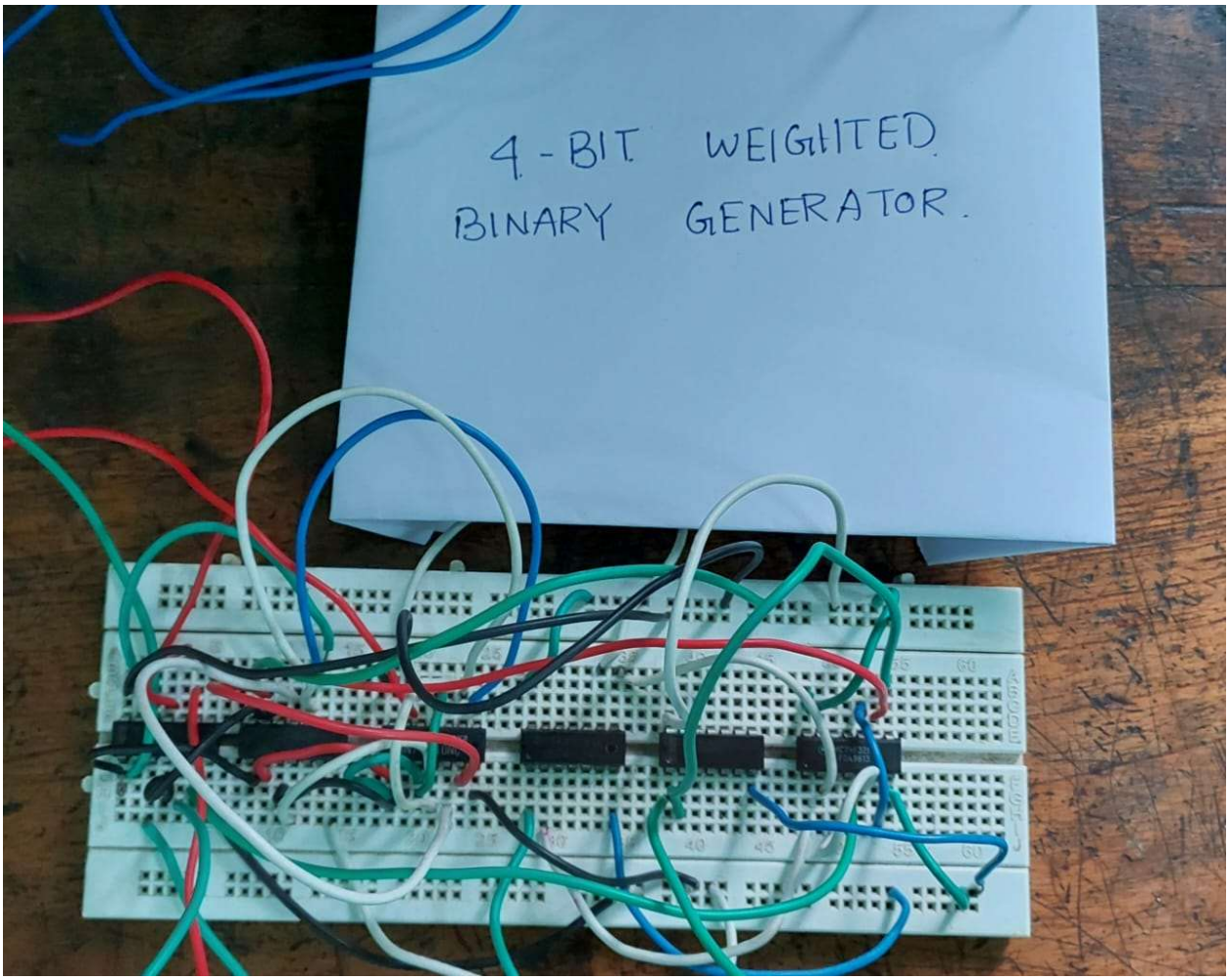
P(X=1)=12/16=3/4



The Picture above is 4 bit stochastic number generator circuit (Comparator based).

The Hardware implementation consists of these IC's

7486 ,7485 ,7474 ,7404 , bread board, digital simulator and connecting wires.



The Above picture is 4 Bit Weighted Binary stochastic number Generator.

The Hardware implementation consists of these IC's

7408,7486 ,7432 ,7474 ,7404 , bread board, digital simulator and connecting wires.

6.2.2 Stochastic number generator FPGA Implementation

In Vivado we have implemented a N_bit SNG which is a comparator based SNG.

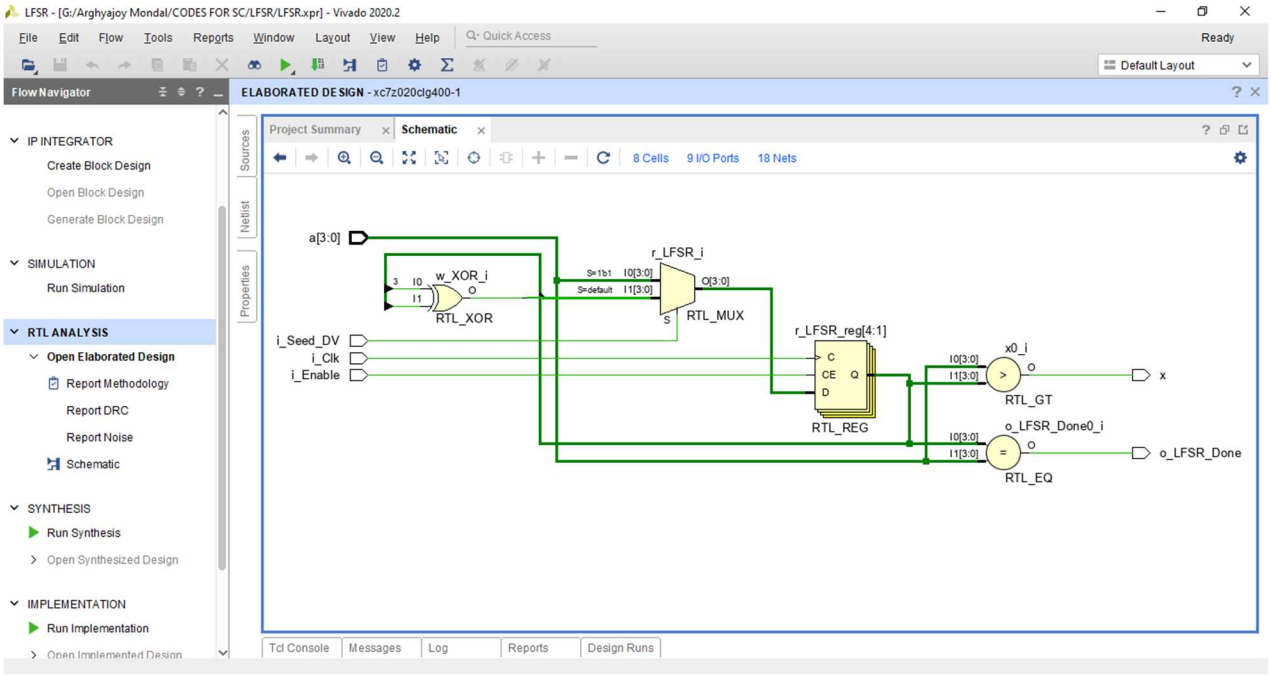
The source code for the N Bit SNG is as follows:

```
-- Company: JADAVPUR UNIVERSITY
-- Engineer: ARGHYAJOY MONDAL
--
-- Create Date: 07/13/2022 03:57:05 PM
-- Design Name:
-- Module Name: n_BIT_LFSR - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity LFSR is
  generic (
    g_Num_Bits : integer := 4
    );
  port (
    i_Clk    : in std_logic;
    i_Enable : in std_logic;

    i_Seed_DV   : in std_logic;
    a : in std_logic_vector(g_Num_Bits-1 downto 0);
```

```
    x : out std_logic;

    o_LFSR_Done : out std_logic
    );
end entity LFSR;

architecture RTL of LFSR is

  signal r_LFSR : std_logic_vector(g_Num_Bits downto 1) := (others => '0');
  signal w_XOR : std_logic;
   signal b : std_logic_vector(g_Num_Bits-1 downto 0);
   signal i_seed_data :  std_logic_vector(g_Num_Bits-1 downto 0);



begin

  -- Purpose: Load up LFSR with Seed if Data Valid (DV) pulse is detected.
  -- Othewise just run LFSR when enabled.
  p_LFSR : process (i_Clk) is
  begin
    if rising_edge(i_Clk) then
      if i_Enable = '1' then
        if i_Seed_DV = '1' then
          r_LFSR <= a;
        else
          r_LFSR <= r_LFSR(r_LFSR'left-1 downto 1) & w_XOR;
        end if;
      end if;
    end if;
  end process p_LFSR;

  g_LFSR_3 : if g_Num_Bits = 3 generate
    w_XOR <= r_LFSR(3) XOR r_LFSR(2);
  end generate g_LFSR_3;

  g_LFSR_4 : if g_Num_Bits = 4 generate
    w_XOR <= r_LFSR(4) XOR r_LFSR(3);
  end generate g_LFSR_4;
```

```
  g_LFSR_5 : if g_Num_Bits = 5 generate
    w_XOR <= r_LFSR(5) XOR r_LFSR(3);
  end generate g_LFSR_5;

  g_LFSR_6 : if g_Num_Bits = 6 generate
    w_XOR <= r_LFSR(6) XOR r_LFSR(5);
  end generate g_LFSR_6;

  g_LFSR_7 : if g_Num_Bits = 7 generate
    w_XOR <= r_LFSR(7) XOR r_LFSR(6);
  end generate g_LFSR_7;

  g_LFSR_8 : if g_Num_Bits = 8 generate
    w_XOR <= r_LFSR(8) XOR r_LFSR(6) XOR r_LFSR(5) XOR r_LFSR(4);
  end generate g_LFSR_8;



  b <= r_LFSR(r_LFSR'left downto 1);
  o_LFSR_Done <= '1' when r_LFSR(r_LFSR'left downto 1) = a else '0';
   i_seed_data <= a;

  process(a,b)
   begin
    if ( a > b) then
    x <= '1';
    else
    x <= '0';
    end if;
    end process;


end architecture RTL;
```

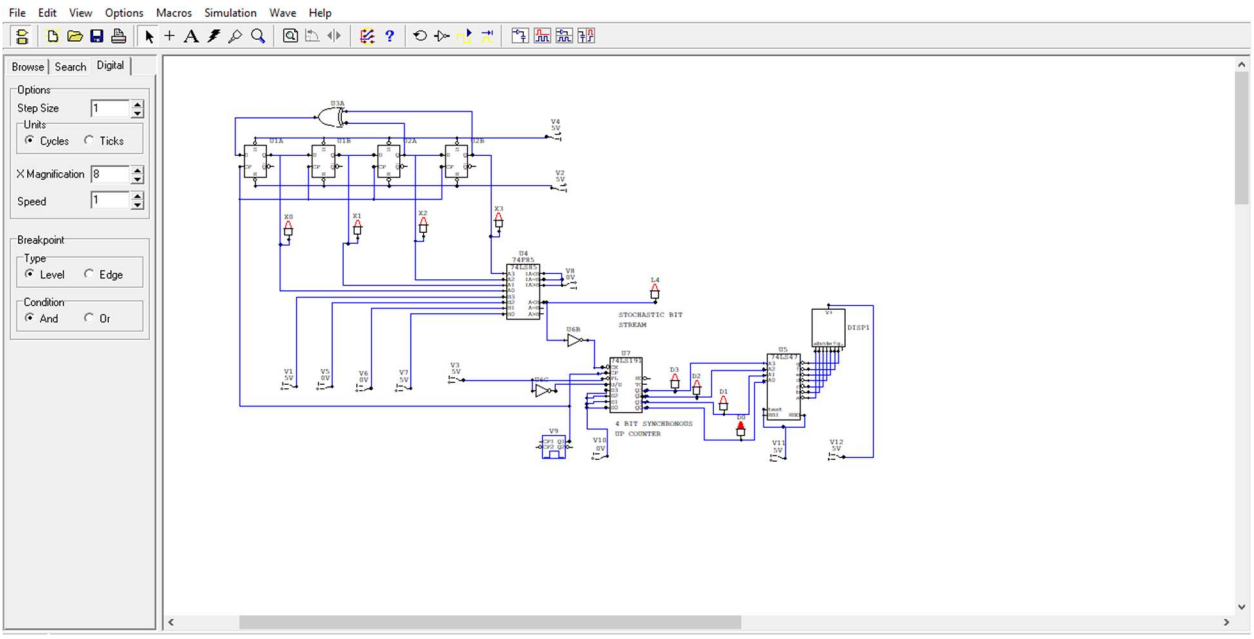
The RTL schematic which consists of both SNG is as follows:



6.3 De-Randomizer Unit

6.3.1 De-Randomizer Unit Circuit maker Implementation

Here In this thesis we have implemented a 4-bit SN to binary circuit in circuit maker 2000
The circuit maker 2000 implementation of a 4-bit SN to binary circuit is as follows:



6.3.1 De-Randomizer Unit Circuit maker Implementation

In Vivado we have implemented a N Bit De Randomizer unit which converts the SN generated from a comparator based SNG.

The source code for the N Bit De Randomizer is as follows:

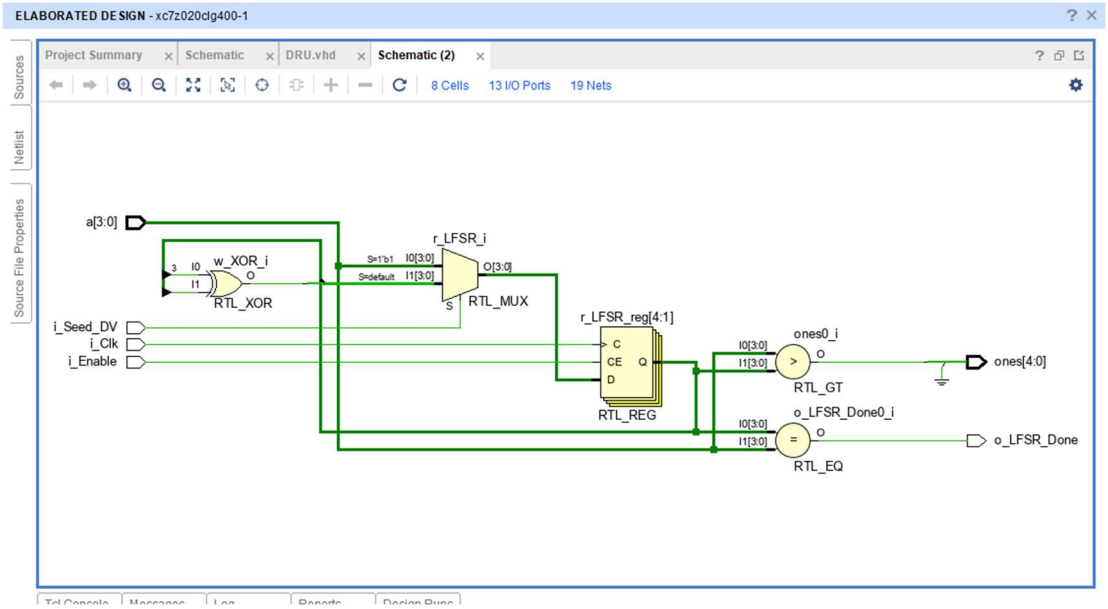
```
----------------------------------------------------------------------------------
-- Company: JADAVPUR UNIVERSITY
-- Engineer: ARGHYAJOY MONDAL
--
-- Create Date: 11/08/2022 08:54:18 PM
-- Design Name:
-- Module Name: DRU - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
-- Revision 0.01 - File Created
-- Additional Comments:
```

```
--
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity LFSR is
  generic (
    g_Num_Bits : integer := 4
    );
  port (
    i_Clk    : in std_logic;
    i_Enable : in std_logic;

    i_Seed_DV   : in std_logic;
    a : in std_logic_vector(g_Num_Bits-1 downto 0);


    ones : out  STD_LOGIC_VECTOR (4 downto 0);

    o_LFSR_Done : out std_logic
    );
end entity LFSR;

architecture RTL of LFSR is

  signal r_LFSR : std_logic_vector(g_Num_Bits downto 1) := (others => '0');
  signal w_XOR : std_logic;
   signal b : std_logic_vector(g_Num_Bits-1 downto 0);
   signal i_seed_data :  std_logic_vector(g_Num_Bits-1 downto 0);
    signal x: std_logic;


begin

  -- Purpose: Load up LFSR with Seed if Data Valid (DV) pulse is detected.
  -- Othewise just run LFSR when enabled.
  p_LFSR : process (i_Clk) is
  begin
    if rising_edge(i_Clk) then
```

```
    if i_Enable = '1' then
      if i_Seed_DV = '1' then
        r_LFSR <= a;
      else
        r_LFSR <= r_LFSR(r_LFSR'left-1 downto 1) & w_XOR;
      end if;
    end if;
  end if;
end process p_LFSR;

g_LFSR_3 : if g_Num_Bits = 3 generate
  w_XOR <= r_LFSR(3) XOR r_LFSR(2);
end generate g_LFSR_3;

g_LFSR_4 : if g_Num_Bits = 4 generate
  w_XOR <= r_LFSR(4) XOR r_LFSR(3);
end generate g_LFSR_4;

g_LFSR_5 : if g_Num_Bits = 5 generate
  w_XOR <= r_LFSR(5) XOR r_LFSR(3);
end generate g_LFSR_5;

g_LFSR_6 : if g_Num_Bits = 6 generate
  w_XOR <= r_LFSR(6) XOR r_LFSR(5);
end generate g_LFSR_6;

g_LFSR_7 : if g_Num_Bits = 7 generate
  w_XOR <= r_LFSR(7) XOR r_LFSR(6);
end generate g_LFSR_7;

g_LFSR_8 : if g_Num_Bits = 8 generate
  w_XOR <= r_LFSR(8) XOR r_LFSR(6) XOR r_LFSR(5) XOR r_LFSR(4);
end generate g_LFSR_8;



b <= r_LFSR(r_LFSR'left downto 1);
o_LFSR_Done <= '1' when r_LFSR(r_LFSR'left downto 1) = a else '0';
 i_seed_data <= a;
```

```
 process(a,b)
  begin
   if ( a > b) then
   x <= '1';
   else
   x <= '0';
   end if;
   end process;
 process(x)

 variable count : unsigned (4 downto 0) := "00000";
  begin
   count := "00000";   --initialize count variable.

      if(x = '1') then --check if the bit is '1'
         count := count + 1; --if its one, increment the count.
      end if;

   ones <= std_logic_vector(count);    --assign the count to output.
end process;

end architecture RTL;
```

The RTL schematic which consists of N BIT De-Randomizer is as follows:



CHAPTER 7 Hardware implementation of stochastic combinational circuits

Here this chapter consists of some of the combinational stochastic circuits, so we have discuss about it in detail and have to implemented it in circuit maker 2000.

So, the combinational circuits we are going to discuss are

7.1 Scaled adder
7.2 Multiplier
7.3 Subtracter

7.1 Scaled adder

In stochastic computing we won't get perfect addition just by using OR gate. Since stochastic numbers are treated as probabilities, they fall naturally into the interval [0,1]. This makes the normal add operation inconvenient because the sum of two numbers from [0,1] lies in [0,2].[2].

Because of this reason we need to perform scaled addition. To accomplish this scaled adder which is implemented by a 2:1 MUX.

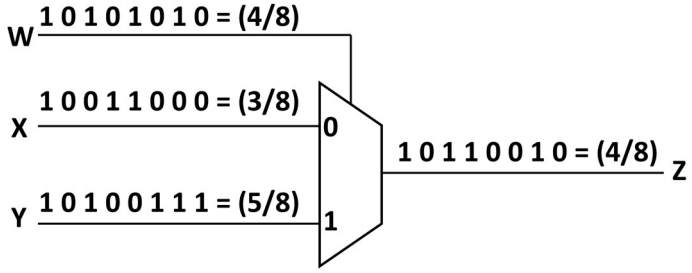
Suppose we applied to its data inputs N1 and N2, a two-way multiplexer can compute the sum of two stochastic numbers, p(X) = x and p(Y) = y. A third number with the constant value p(W) = w = 1/2 is also needed and is applied to the multiplexer's third (select) input; this can be provided by a (pseudo) random number generator.

Here, the correlation between the two input numbers should be zero.[2]

The probability of a 1 appearing at the output z = p(Z) is then equal to the probability of 1 at p(W) multiplied by the probability of 1 at p(X), plus the probability of 0 at p(W) multiplied by the probability of 1 at p(Y).

More formally,

$$z = xw + (1 - w)y = (x + y) / 2$$



Here, W effectively scales the sum by 1/2

In the figure, we can see that **x = 3/8** is added with **y = 5/8**. With the help of the 3rd input, which is a random bit-stream of **w = 4/8**, the multiplier performs addition of **x** and **y** and produces the output, which is **z = 4/8**.

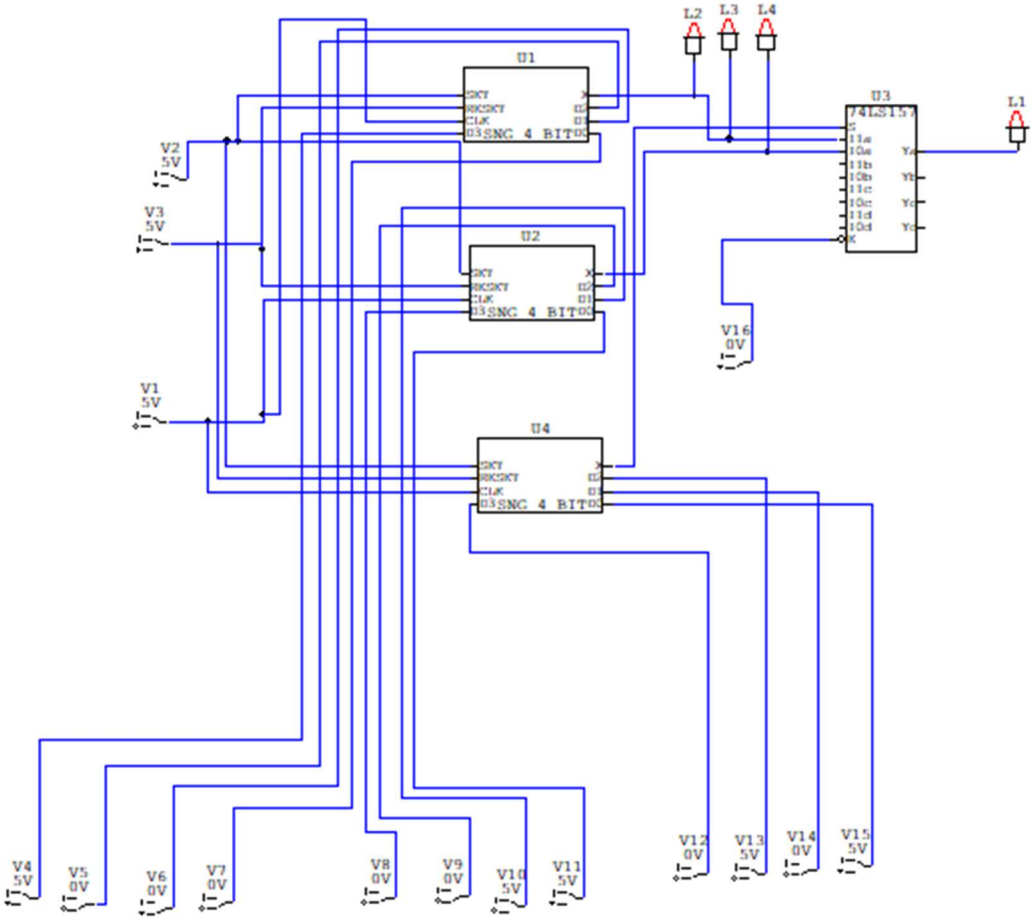
z = (x + y) / 2

or, z = (3/8 + 5/8) / 2

or, z = 4/8

In this thesis we have implemented a 4 bit scaled circuit in circuit maker 2000

The circuit maker 2000 implementation of a 4-bit scaled adder circuit is as follows:



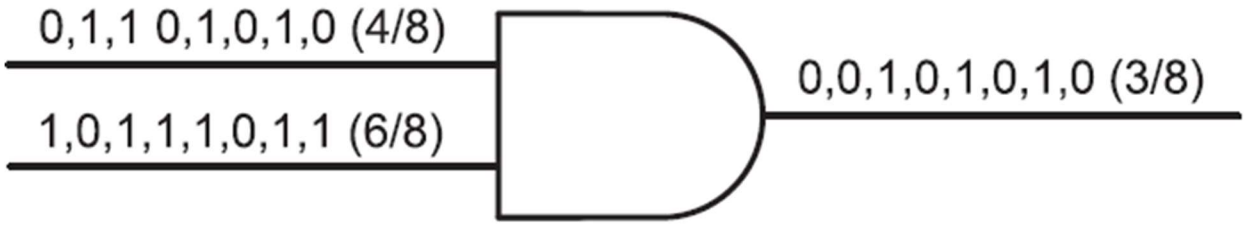
7.2 Multiplier

In unipolar coding format stochastic number multiplication can be obtained by just using a AND gate but the condition is that the correlation between the two stochastic numbers should be zero.

So let suppose two stochastic bit streams X and Y have probability of getting a 1's in their bit stream is P(X) and P(Y).

Then the probability of getting a 1 in the output bit stream of the AND gate Z will be

P(Z)=P(X).P(Y)



Let suppose two independent random bit streams X and Y as inputs to a two-input AND gate where,

$$p(X) = x = 4/8$$

$$p(Y) = y = 6/8$$

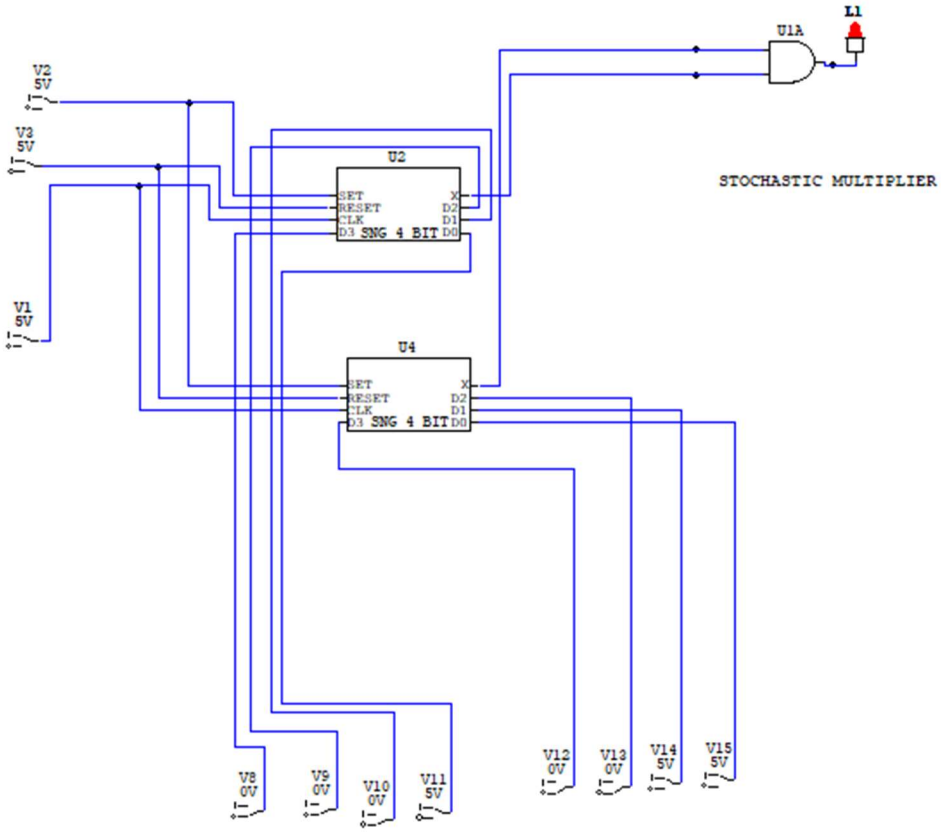
The output random bit stream is then given by,

$$z = p(Z) = p(X , Y) = p(X)\, p(Y) = xy = 3/8$$

Thus, one AND gate can perform stochastic multiplication.

In this thesis we have implemented a 4 bit multiplier circuit in circuit maker 2000

The circuit maker 2000 implementation of a 4 bit multiplier circuit is as follows:



7.3 Subtracter

Subtraction can be implemented by a single XOR.

But the condition is both the stochastic numbers has to be highly correlated.

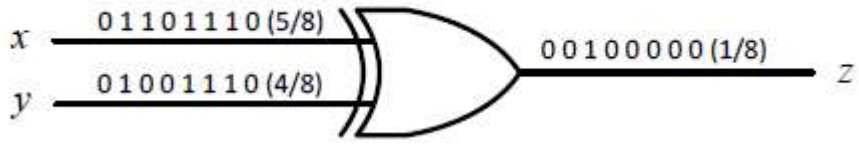
Here, inputs p(X) and p(Y) (of x and y, respectively) are highly correlated.

Because of the maximum overlapping of 0s and 1s due to high correlation, the probability of receiving two 1s (or two 0s) on X and Y is min (p (X), p (Y)) (or min (1 - p (X), p (Y))), implying that the probability of receiving different values on x and y is |p (X) - p(Y)|.

If stochastic numbers p(X) and p(Y) are uncorrelated, then XOR implements an entirely different function.

Function of XOR gate (highly correlated inputs) is

$$\mathbf{z = p(Z) = p(X)\,(1 - p(Y)) + p(Y)\,(1 - p(X)) = |p(X) - p(Y)| = |x - y|}$$



Here In this figure we can see that, **p(X) = 2/8,** and **p(Y) = 7/8** is subtracted and after the subtraction, we get **p(Z) = 1/8**

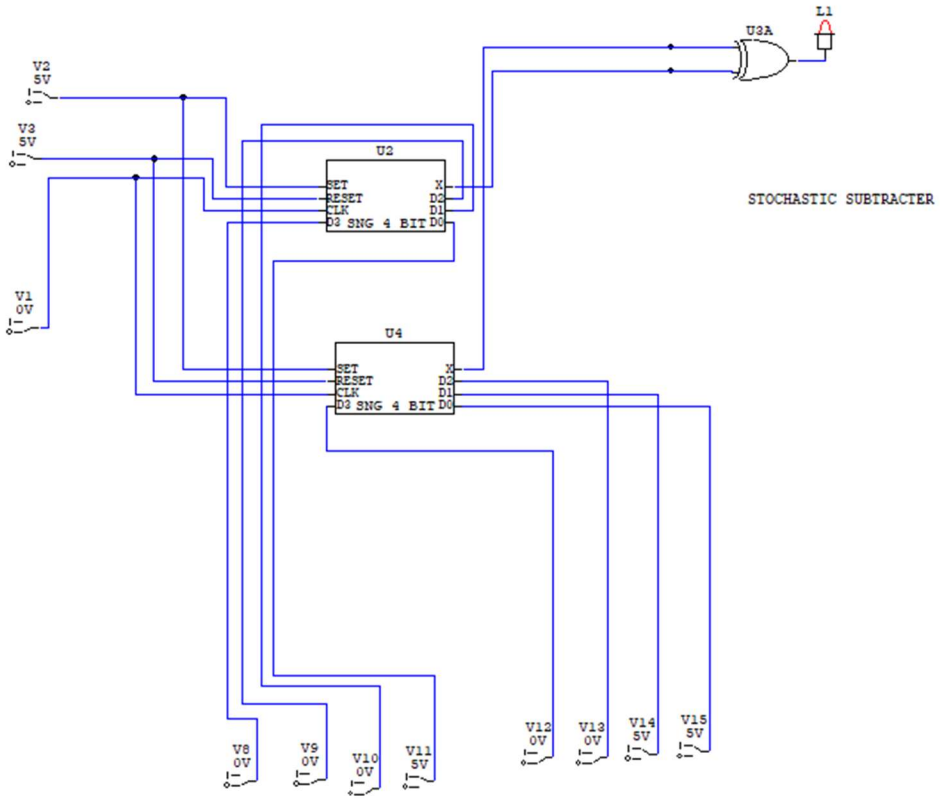
$$\mathbf{p(Z) = |p(X) - p(Y)|}$$

$$\mathbf{or,\ p(Z) = |2/8 - 7/8|}$$

$$\mathbf{or,\ p(Z) = 5/8}$$

In this thesis we have implemented a 4 bit subtracter circuit in circuit maker 2000

The circuit maker 2000 implementation of a 4-bit subtracter circuit is as follows:



CHAPTER 8 Stochastic greatest common divisor

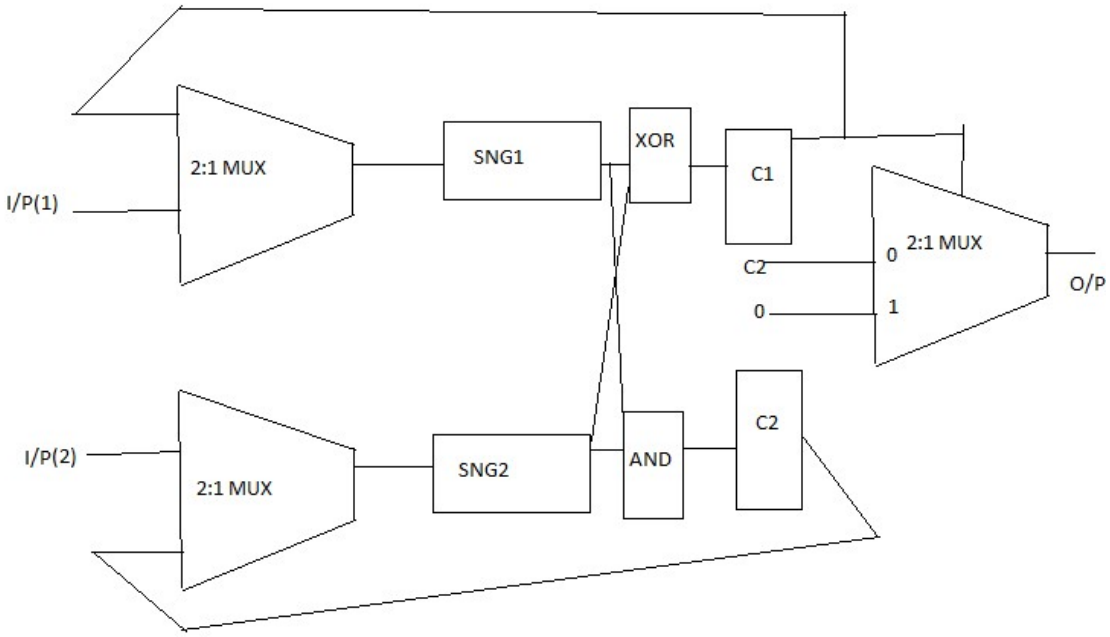
This chapter consists of a stochastic GCD circuit, some theory on it and its hardware implementation and later simulation of the same.

8.1 Theoretical Background

In elementary arithmetic, the greatest common divisor is used to simplify expressions by reducing the size of numbers involved. Greatest common divisor (GCD) of given numbers is the largest number that divides all of the given numbers without leaving any remainder.

GCD computation is a central task in computer algebra, in particular when computing over rational numbers or over modular integers [11] . Greatest Common Divisor can be found using many methods.

GCD (a, b) = max {kЄZ : k|a ^ k|b}[12]



This is a block diagram of the implemented stochastic GCD circuit.

The operation of the stochastic GCD circuit is really simple.

In this stochastic GCD circuit we are taking into consideration that both the inputs to the stochastic GCD circuit are highly correlated thus the inputs of the stochastic GCD circuit has maximum stochastic Correlation Coefficient i.e +1.

So that XOR can be used as a subtracter and the AND gate can be used to find the minimum of two numbers.

Working principle of the implemented GCD circuit:

So lets suppose we have input (1)=7 and input(2)=2

So the algorithm as follows

First iteration: Absolute subtraction of in(1) and in(2)(7-2=5)

Min of the two numbers(min(7,2)=2)

Second iteration: Absolution subtraction of the iteration 1 answer

and min of 2(5-2=3)

Min of the two numbers(min(5,2)=2)

Third iteration: Absolution subtraction of the iteration 2 answer

and min of 2(3-2=1)

Min of the two numbers(min(3,2)=2)

Fourth iteration: Absolution subtraction of the iteration 3 answer

and min of 2(2-1=1)

Min of the two numbers(min(1,2)=1)

Fourth iteration: Absolution subtraction of the iteration 4 answer

and min of 2(1-1=0)

Min of the two numbers(min(1,1)=1)

So when we get C1 counter value as zero , the counter 2 or C2 value is the GCD of the these 2 numbers.

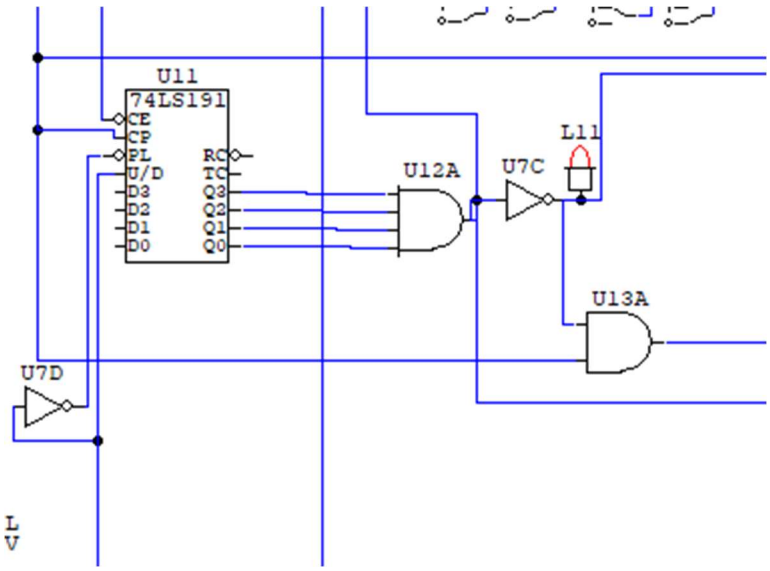
This iteration cycle will be continued till the counter 2 value will be zero.

8.2 Hardware Implementation of Stochastic GCD circuit

Hardware implementation of the stochastic GCD circuit consist of SNG's, counters, multiplexers, registers, XOR and AND gates.

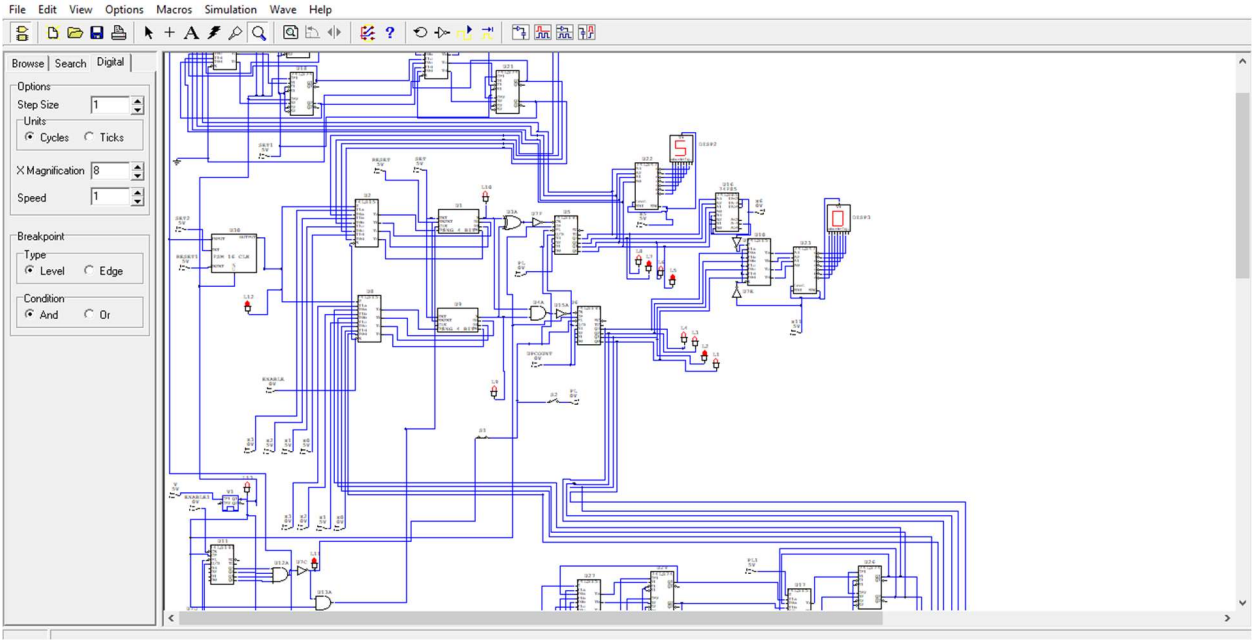
But we have to keep in mind that during the simulation on every 16'th clock pulse the SNG will remain idle so there is an deficit of a single clock pulse so have proposed an FSM based model by which have we can fulfil that particular single clock pulse or the 16'th clock pulse.

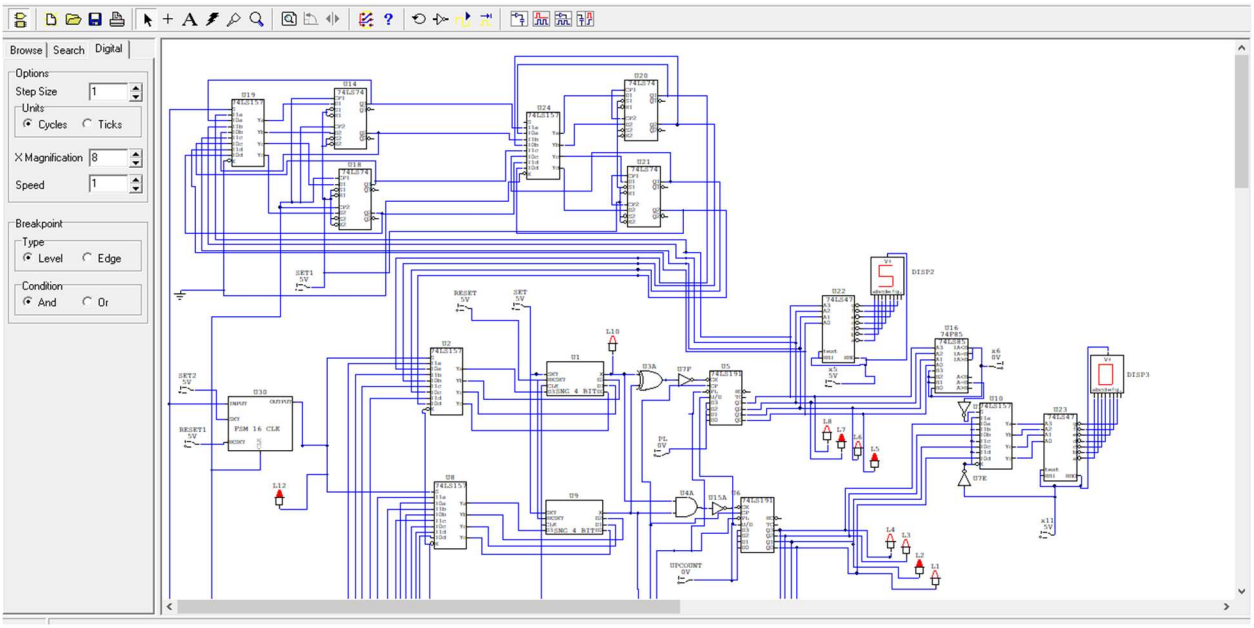
Our proposed method is as follows



Here 74191 is a 4 bit counter which counts upto 16 . Now every 16 th clock pulse the and operation of the all bits o/p is always 1. In the other words on 16 th clock pulse the NAND operation of the counter o/p always produce 0. Now if we do AND operation with clock pulse and send it to the 16 th clock pulse then SNG will skip the 16 th clock pulse and work as usual.

Now we have implemented a 4-bit stochastic GCD circuit in circuit maker 2000 As follows:





In Vivado we have implemented a single iteration of stochastic GCD circuit which is a comparator based SNG.

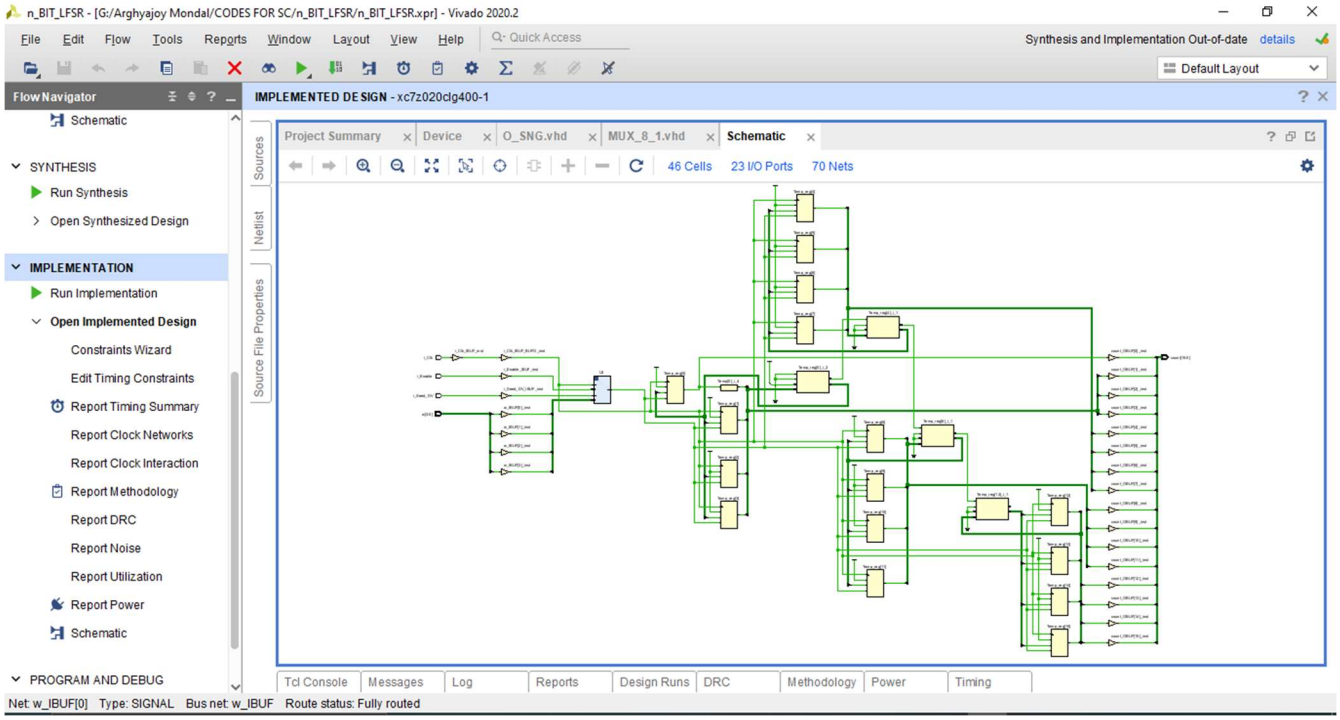
The source code for the stochastic GCD is as follows:

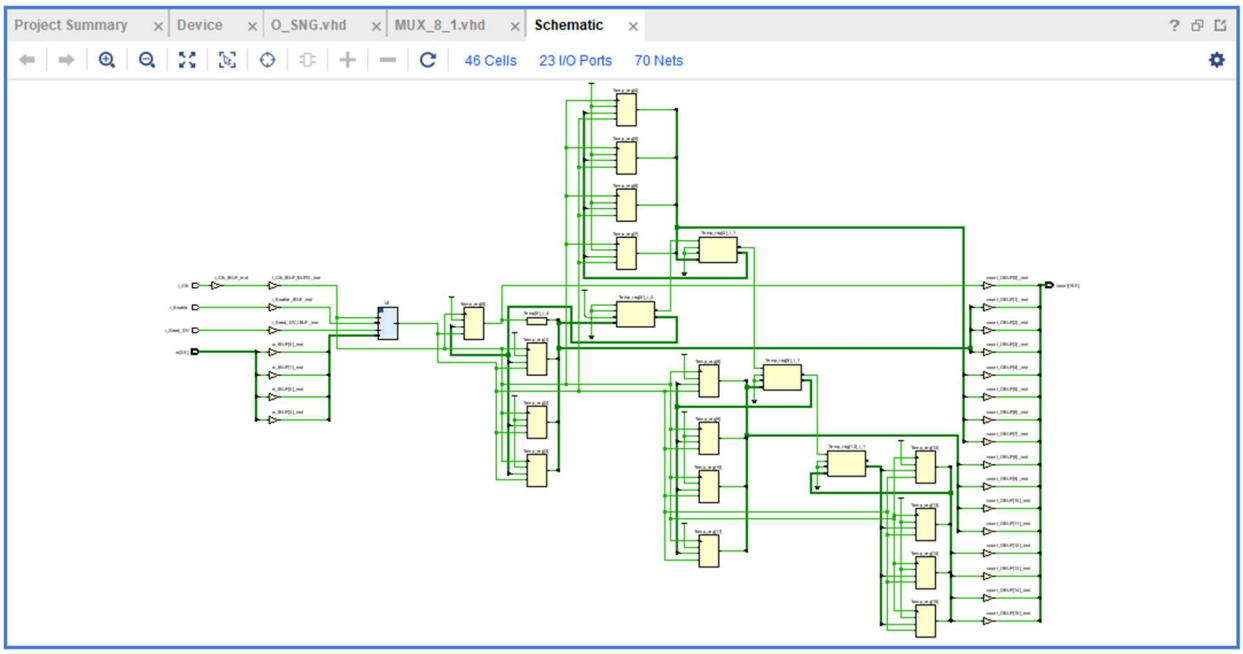
```
----------------------------------------------------------------------------------
-- Company:  JADAVPUR UNIVERSITY
-- Engineer: ARGHYAJOY MONDAL
-- Create Date: 05/18/2022 09:59:56 AM
-- Design Name:
-- Module Name: O_SNG - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1 is
  port (w1, x1, y1, z1, s : in std_logic;
     f : out std_logic_vector(3 downto 0));
end mux2to1;

architecture behaviour of mux2to1 is
begin
  process (w1, x1, y1, z1, s)
  begin
    if s = '1' then
     f(0) <= w1;
      f(1) <= x1;
      f(2) <= y1;
      f(3) <= z1;
    else
      f<= (OTHERS =>'0');
    end if;
  end process;
```

```
end behaviour;
----------------------------------------------------------------------------------------------------
------------------------------------------------------

 library ieee;
   use ieee.std_logic_1164.all;
   use IEEE.std_logic_signed.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity O_SNG is
generic (
   g_Num_Bits : integer := 4
   );
 port (w: in std_logic_vector(3 downto 0);--input 1 at the 2:1 mux
      i_Clk    : in std_logic;
      i_Enable : in std_logic;
   i_Seed_DV   : in std_logic;

    count: out std_logic_vector(2**(g_Num_Bits)-1 downto 0));

end O_SNG;

architecture Behavioral of O_SNG is

signal f : std_logic_vector(3 downto 0);
signal y: std_logic ;
signal Temp: std_logic_vector(2**(g_Num_Bits)-1 downto 0);

component mux2to1 is
 port (w1, x1, y1, z1, s : in std_logic;
```

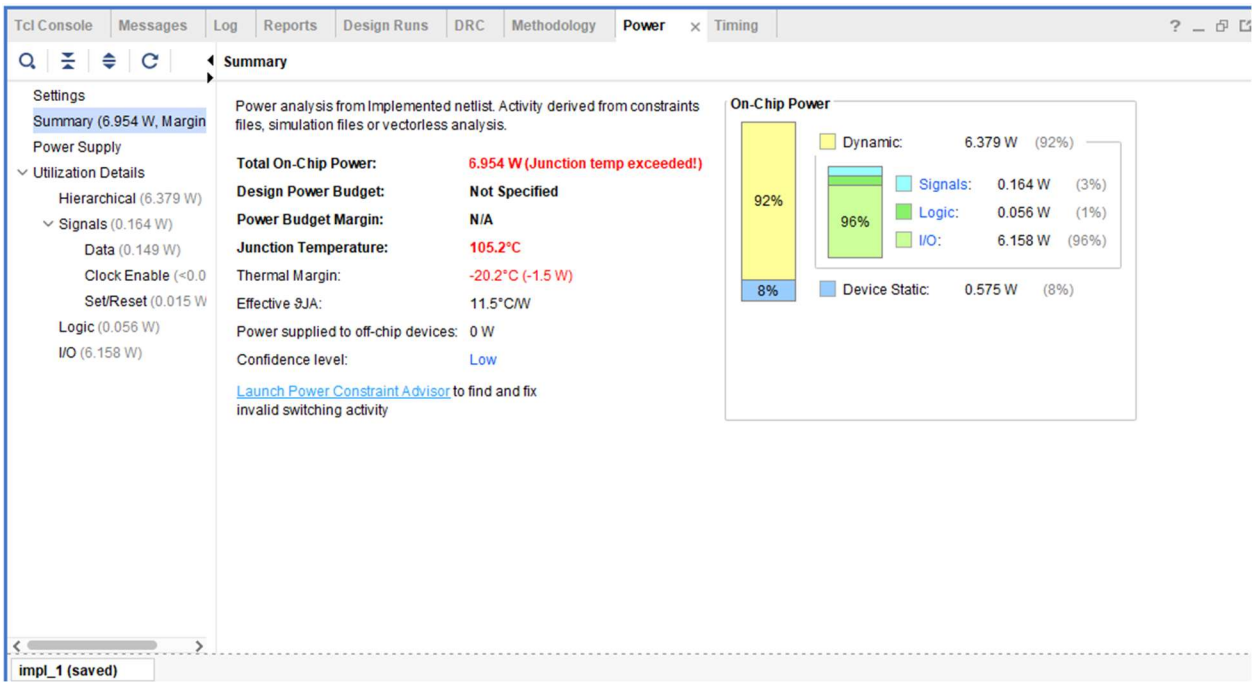
```
    f : out std_logic_vector(3 downto 0));
    end component mux2to1;

    component LFSR is
     port (
   i_Clk    : in std_logic;
   i_Enable : in std_logic;

   i_Seed_DV   : in std_logic;
   a : in std_logic_vector(g_Num_Bits-1 downto 0);


   x : out std_logic;

   o_LFSR_Done : out std_logic
   );
   end component LFSR;

begin
U1:mux2to1 port map(w1=>w(3),x1=>w(2),y1=>w(1),z1=>w(0),s=>'1',f=>f);
U2:LFSR port
map(i_Clk=>i_Clk,i_Enable=>i_Enable,i_Seed_DV=>i_Seed_DV,a=>f,x=>y);

process(i_Clk,y)
begin
 if i_Clk'EVENT AND i_clk='1' then
   if y = '1'  then
      Temp <= Temp + "000000000000001";
     else
     Temp <= (OTHERS =>'0');
    end if;
    end if;
    end process;
    count <= Temp;

end Behavioral;
```

The RTL schematic of Stochastic GCD circuit is as follows:





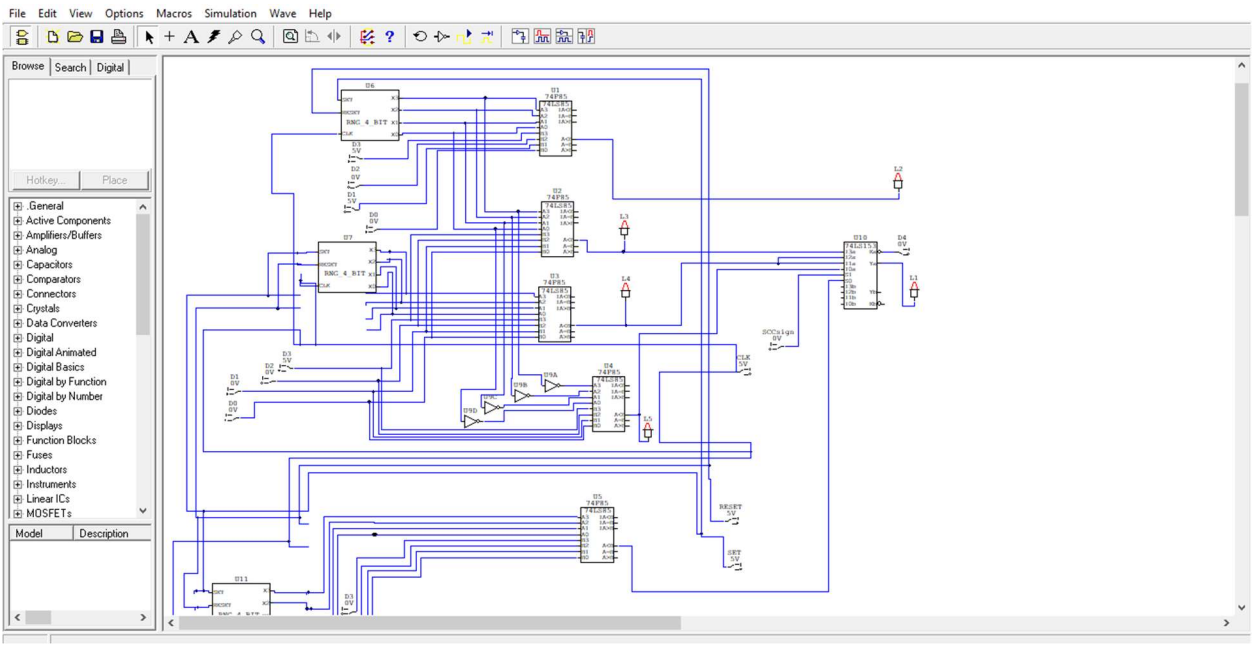
Power analysis of the Stochastic GCD circuit is as follows:



CHAPTER 9: GENERATION OF THE SN'S WITH A SPECIFIED SCC:

In the previous chapter 4 we already got the idea the of how stochastic numbers and circuits are heavily affected by the change of stochastic correlation coefficient.
In this chapter we will going to see the implementation of a stochastic number generator with a specified SCC.

Now the circuit maker implementation of Stochastic number generator with a specified SCC is as follows.



This circuit is influenced by the existing circuit of a stochastic number generator which is presented by Alaghi& Hayes[13].

By simulating this circuit we got results which matches the expected output.

Simulation Results:

CASE 1:

For SCCsign= 0 SCCmagn=0 and the Bit stream is 1010

X	0	0	1	1	1	1	1	1	1	0	0	1	0	1	0	0
Y	1	1	1	0	0	0	1	0	1	1	1	0	0	1	1	1

P(X)=9/16 and P(Y)=10/16

CASE 2:

For SCCsign= 0 SCCmagn=1 and the Bit stream is 1010

X	0	0	1	1	1	1	1	1	1	0	1	0	1	1	0	0
Y	0	0	1	1	1	1	1	1	1	0	1	0	0	1	0	0

P(X)=10/16 and P(Y)=9/16

CASE 3:

For SCCsign= 1 SCCmagn=0 and the Bit stream is 1010

X	0	1	1	1	1	1	1	1	0	0	1	0	1	0	0	1
Y	0	1	1	1	1	1	1	1	0	0	1	0	1	0	0	1

P(X)=10/16 and P(Y)=10/16

CASE 4:

For SCCsign= 1 SCCmagn=1 and the Bit stream is 1010

X	0	1	1	1	1	1	1	1	0	0	1	0	1	0	0	1
Y	0	1	1	1	1	1	1	1	0	0	1	0	1	0	0	1

P(X)=10/16 and P(Y)=10/16

CHAPTER 10: Conclusion and future scope for research

The SNG we have used in most of the circuits is using 4 bits as input binary stream but if the number of bits increase then the accuracy of the generated SN will be increased significantly.

From this thesis we mainly got the idea of a stochastic GCD circuit and its implementation on the FPGA platform and also it shows the on-chip power dissipation and other parameters like area, latency which we got by generating netlist on the VIVADO 2020.2.

During source code adding we must add the board files of the respective FPGA otherwise that will create issues during the bit stream generation and importing the files into FPGA board.

In chapter 9 the SNG with specific SCC simulation shows us that in order to get maximum correlation among the stochastic number we must apply SCCsign=0 and SCCmagn=+1 and in order to get uncorrelated stochastic bit streams we must apply SCCsign=0 and SCCmagn=0.

From elementary arithmetic to the complex neural networks, the greatest common divisor is used to simplify expressions by reducing the size of numbers involved.
The stochastic GCD circuit is implemented in this thesis has several advantages over other architectures but still its constrained by the fact that it still uses a SNG to generate the stochastic number which consists the LFSR which is a major drawback in circuit latency, so there is so much work needs to be done to remove these limitations.

CHAPTER11 References

[1] A.Alaghi, John P. Hayes," Survey of Stochastic Computing", ACM Transactions on Embedded Computing Systems, Vol. 12, No. 2s, Article 92, 2013.

[2] B.R.Gaines, "Stochastic Computing Systems, Advances in Information Systems Science," J.F.Tou, ed., vol.2, chapter 2, pp.-37-172, New York Plenum 1999.

[3] B.D.Brown and H.C.Card, "Stochastic neural computation, I, Computational elements," IEEE Trans. Comput., vol. 50, no. 9, pp. 891-905, Sep. 2001

[4] W.Qian and M.D.Reidel, "The Synthesis of Robust Polynomial Arithmetic withStochastic Logic," Proc. 45th ACM/IEEE Design Automation Conf., pp. 648-653, 2008

[5] M.H.Najafi, S.Jamali-Zaraveh, D.Lilja, M.Reidel, K.Bazargan and R.Harjani, "Time-Encoded Values for Highly Efficient Stochastic Circuits," IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol.25, pp.1644-165, January 2017

[6] S.-J. Min, E.-W. Lee and S.-I. Chae, "A Study on the Stochastic Computation Using the Ratio of One Pulses and Zero Pulses," Proc. Intl. Symp. Circuits and Systems(ISCAS), pp. 471–474, 1994.

[7] Goresky, M. and Klapper, A.M. Fibonacci and Galois representations of feedback-with-carry shift registers, IEEE Transactions on Information Theory, Nov 2002, Volume: 48, On page(s): 2826 –2836.

[8] FPGA Implementation of 8, 16 and 32 Bit LFSR with Maximum Length Feedback Polynomial using VHDL.

[9] Armin Alaghi and John P. Hayes, "Fast and Accurate Computation using Stochastic Circuits", Advanced Computer Architecture Laboratory Department of Electrical Engineering and Computer Science, University of Michigan Ann Arbor, MI, 48109, USA

[10]Bahram Dehghan , Naser Parhizgar, "Survey the stochastic computing and probabilistic transfer matrix (PTM) in logic circuits", Department of Electrical

Engineering, College of Engineering, Shiraz Branch, Islamic Azad University, Shiraz, Iran

[11] P. Emeliyanenko, (2010a) A complete modular resultant algorithm targeted for realization on graphics hardware, PASCO 10, pp. 35-43, New York, NY, USA. ACM ;(2010b)

[12] Computational circuit approach to design the greatest common divisor circuits based on methodological analysis and most efficient

[13] Armin Alaghi and John P. Hayes, "Exploiting Correlation in Stochastic Circuit Design", Advanced Computer Architecture Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 48109, USA

[14] Smita Krishnaswamy, George F. Viamontes, Igor L. Markov, and John P. Hayes, "Probabilistic Transfer Matrices in Symbolic Reliability Analysis of Logic Circuits", University of Michigan, Ann Arbor