UFUOMA AYA
SID: 200327306

DATE: 4/20/2020

ENEL 489 PROJECT

ARTIFICIAL NEURAL NETWORK (ANN) BASED ON ROSSLER’S CHAOTIC SYSTEM GENERATOR ON FPGA USING VHDL

Table of contents

1.0 Introduction

Due to the ever-increasing use of the computer in today's world, a powerful and interesting part of Machine Learning is Deep Learning. This is an approach in which computers learn to perform tasks that naturally come to human beings and we can see such technology implemented in driverless vehicles, voice-activated virtual assistants in devices such as smartphones, tablets, TVs, hands-free speakers e.t.c.

With the implementation of neural network architectures and large sets of data, computer models are trained to accomplish human-level performance and accuracy. In this report, we will implement the artificial neural network (ANN) based Rossler's chaotic system generator on FPGA using VHDL.

2.0 Rossler System

This is an autonomous system founded by Otto Rossler. It was developed as the minimum system required to exhibit chaos. The equations below are mathematical representations of the system:

- $\frac{dx}{dt} = -y - z$
- $\frac{dy}{dt} = x + ay$
- $\frac{dz}{dt} = b + (x - c) z$

As shown above, the first and second equations contain linear components which causes oscillation between the x and y variables. The third equation contains only one non-linear component which causes the system to behave in a chaotic manner.

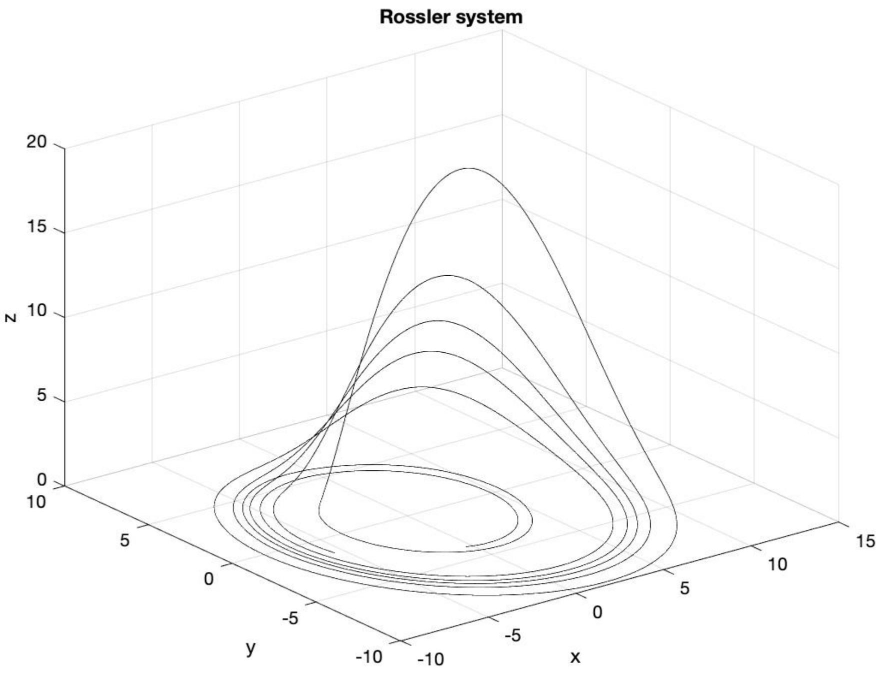


Fig. 1: Rossler system

3.0 Implementation

3.1 RTL Diagram

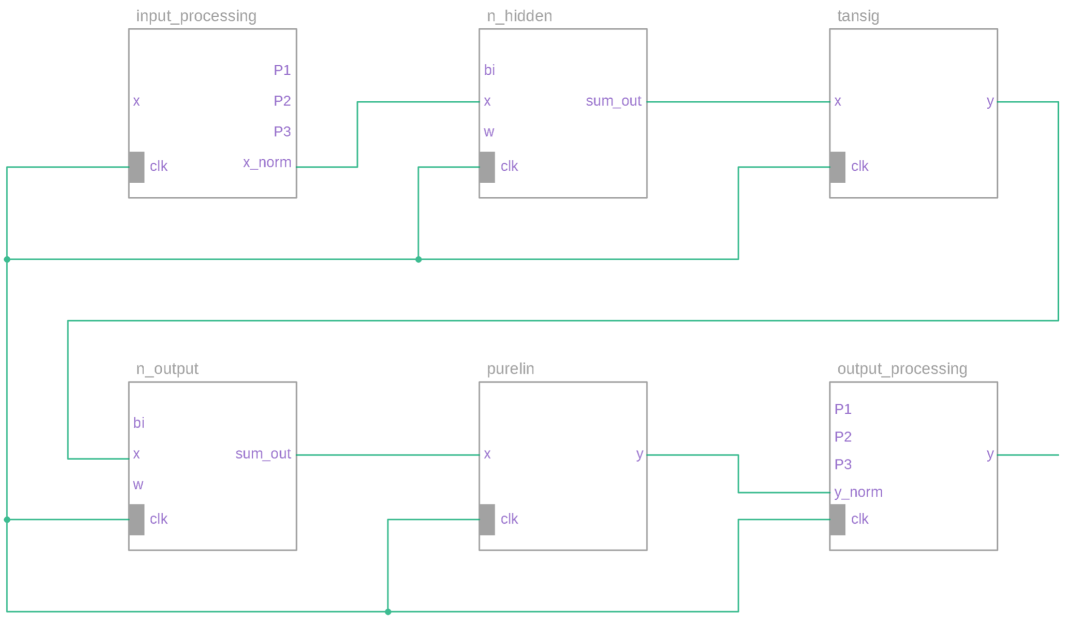


Fig. 2: Generic RTL diagram

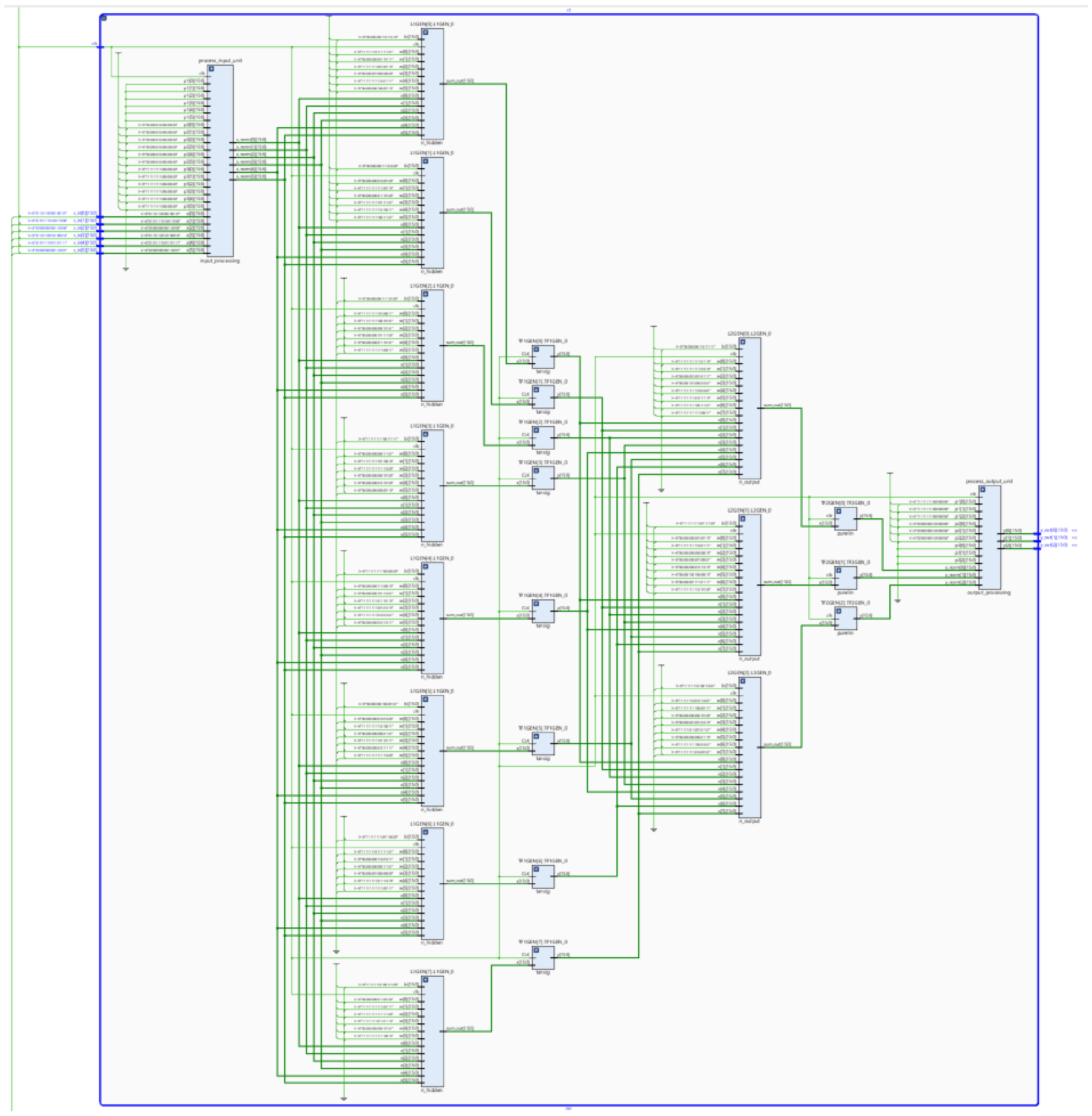


Fig. 3: Elaborated RTL schematic diagram

3.2 Fixed Point Format

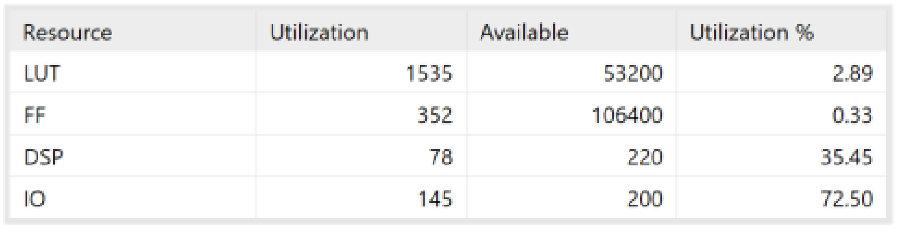
Initially, I chose the 32-bit (Q16.16) fixed point format but while running the implementation, the IO placement turned out to be unfeasible because the number of unplaced terminals (288) were greater than the number of available sites (200). To resolve this issue, I chose the 16-bit (Q4.12) fixed point format which was successfully implemented.

3.3 Network Configuration

With the network configuration files provided, I chose the network configuration file that contains 8 hidden layer neurons while taking into consideration the mean square error. The 8 hidden layer neurons have the smallest mean square error, hence, more accuracy when compared to the other configurations. Also, this network configuration contains 6 inputs and 3 target outputs.

3.4 Report Utilization

Below are information acquired after successfully implementing the neural network design on the Zedboard Zynq Evaluation and Development kit.



Resource	Utilization	Available	Utilization %
LUT	1535	53200	2.89
FF	352	106400	0.33
DSP	78	220	35.45
IO	145	200	72.50

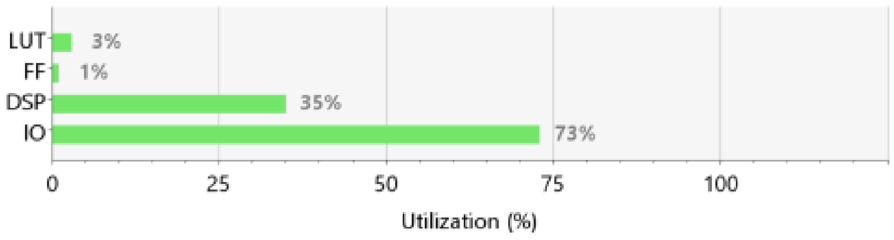
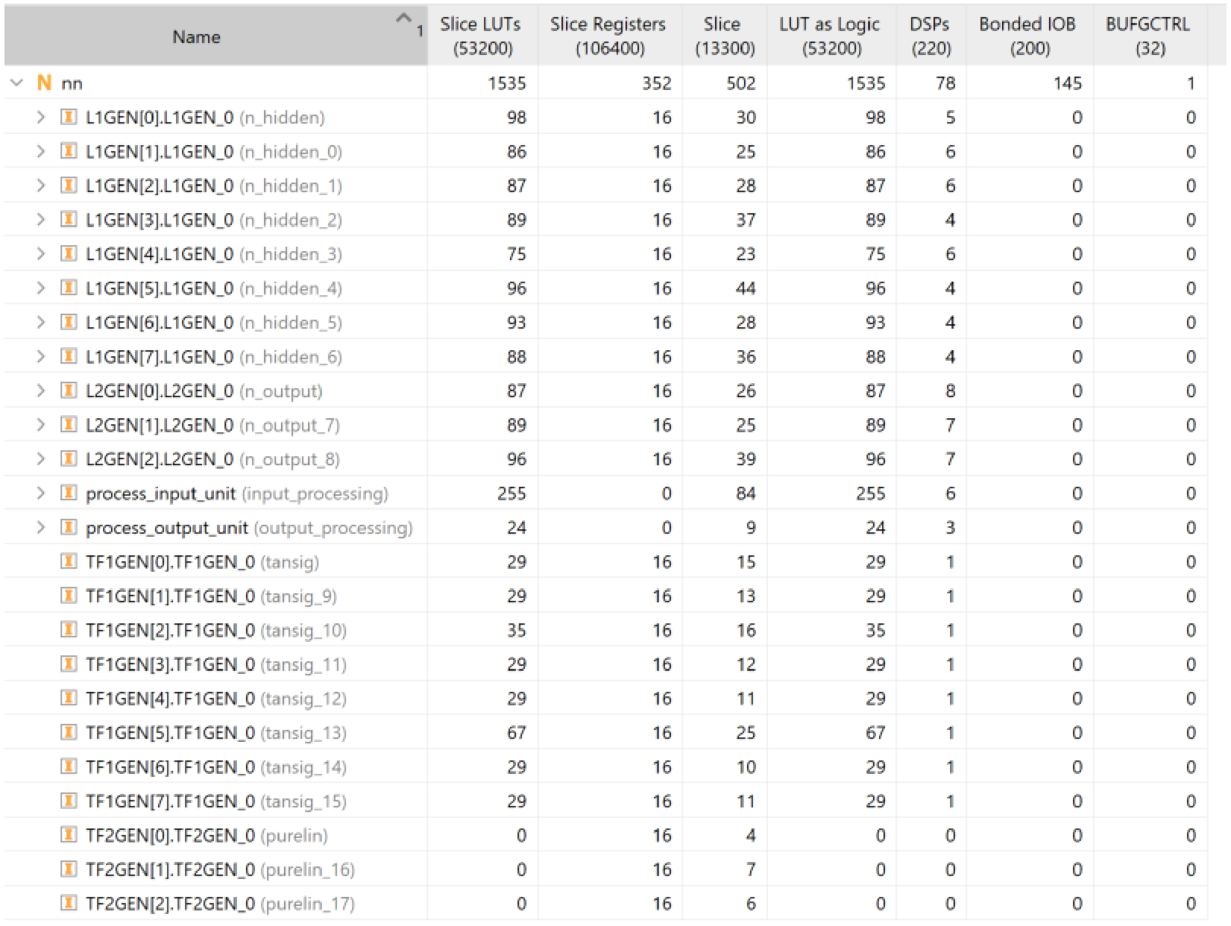


Fig. 4: Utilization summary

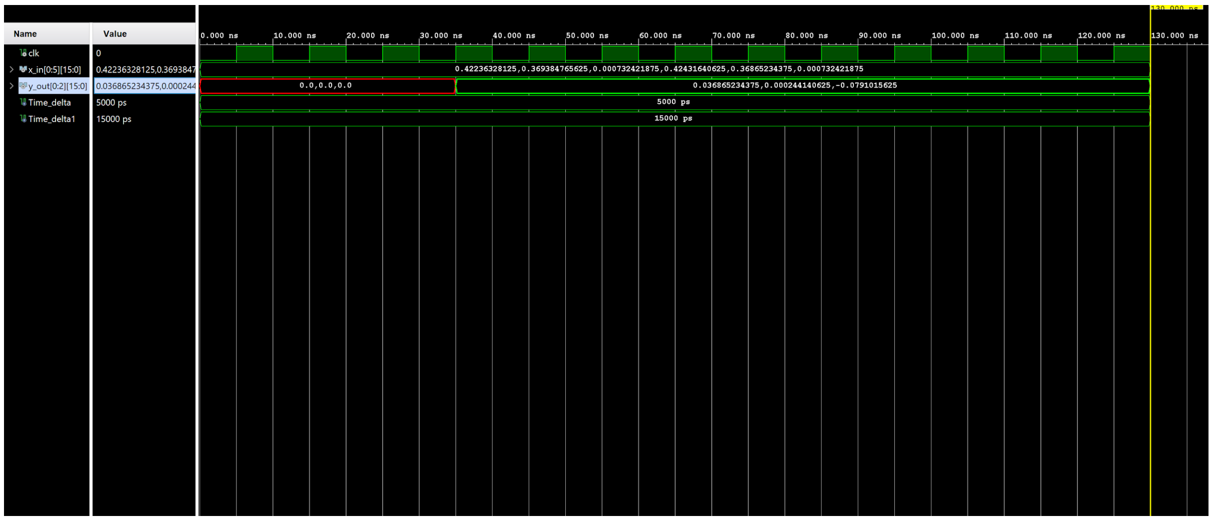


Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	DSPs (220)	Bonded IOB (200)	BUFGCTRL (32)
nn	1535	352	502	1535	78	145	1
L1GEN[0].L1GEN_0 (n_hidden)	98	16	30	98	5	0	0
L1GEN[1].L1GEN_0 (n_hidden_0)	86	16	25	86	6	0	0
L1GEN[2].L1GEN_0 (n_hidden_1)	87	16	28	87	6	0	0
L1GEN[3].L1GEN_0 (n_hidden_2)	89	16	37	89	4	0	0
L1GEN[4].L1GEN_0 (n_hidden_3)	75	16	23	75	6	0	0
L1GEN[5].L1GEN_0 (n_hidden_4)	96	16	44	96	4	0	0
L1GEN[6].L1GEN_0 (n_hidden_5)	93	16	28	93	4	0	0
L1GEN[7].L1GEN_0 (n_hidden_6)	88	16	36	88	4	0	0
L2GEN[0].L2GEN_0 (n_output)	87	16	26	87	8	0	0
L2GEN[1].L2GEN_0 (n_output_7)	89	16	25	89	7	0	0
L2GEN[2].L2GEN_0 (n_output_8)	96	16	39	96	7	0	0
process_input_unit (input_processing)	255	0	84	255	6	0	0
process_output_unit (output_processing)	24	0	9	24	3	0	0
TF1GEN[0].TF1GEN_0 (tansig)	29	16	15	29	1	0	0
TF1GEN[1].TF1GEN_0 (tansig_9)	29	16	13	29	1	0	0
TF1GEN[2].TF1GEN_0 (tansig_10)	35	16	16	35	1	0	0
TF1GEN[3].TF1GEN_0 (tansig_11)	29	16	12	29	1	0	0
TF1GEN[4].TF1GEN_0 (tansig_12)	29	16	11	29	1	0	0
TF1GEN[5].TF1GEN_0 (tansig_13)	67	16	25	67	1	0	0
TF1GEN[6].TF1GEN_0 (tansig_14)	29	16	10	29	1	0	0
TF1GEN[7].TF1GEN_0 (tansig_15)	29	16	11	29	1	0	0
TF2GEN[0].TF2GEN_0 (purelin)	0	16	4	0	0	0	0
TF2GEN[1].TF2GEN_0 (purelin_16)	0	16	7	0	0	0	0
TF2GEN[2].TF2GEN_0 (purelin_17)	0	16	6	0	0	0	0

Fig. 5: Utilization hierarchy

3.5 Timing Simulation

With the implementation of a testbench, I derived the timing simulation for the system shown in the figure below. A propagation delay occurred before the system generated output data. This delay occurred for about 30 ns and it is a result of data moving through the shift registers in the design.



```
Name              Value
clk               0
x_in[0:5][15:0]   0.42236328125,0.3693847
y_out[0:2][15:0]  0.036865234375,0.000244
Time_delta        5000 ps
Time_delta1       15000 ps
```

Fig. 6: Timing simulation diagram

4.0 Limitations

Due to the Covid-19 pandemic, several resources required to the completion of this project were unavailable. Resources such as: Matlab & Simulink application by Mathworks to retrieve data to be implemented into the design and the Zedboard Zynq Evaluation and Development kit to test the implemented design.

5.0 Conclusion

In spite of the limitations due to the Covid-19 pandemic, I was able to successfully synthesize and implement the artificial neural network (ANN) based Rossler's chaotic system generator on FPGA using VHDL. Data to be retrieved from the Matlab & Simulink application were provided in spreadsheet files, hence, the successful implementation design.

References

Chauhan, Nagesh Singh. “Introduction to Artificial Neural Networks(ANN).” *Medium*, Towards Data Science, 10 Oct. 2019, towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9.

THE RÖSSLER SYSTEM, minitorn.tlu.ee/~jaagup/uk/dynsys/ds2/chaos/Rossler/Rossler.html.