

# *Chapter 8*

**FPGA Implementation of  
Soft AODV**

*This chapter has described digital implementation of Artificial Neural Network (ANN), using VHDL programming language. The Very high speed integrated circuit Hardware Description Language (VHDL) is used for programming of ANN architecture. VHDL simplifies the development of complex system of ANNs. FPGA implementation is described for MLP architecture (3 layered 2 inputs and 1 output). AODV parameters adaptively selected using ANFIS for performance optimization of WANET.*

---

Before beginning a hardware implementation of an ANN, a number format (fixed, floating point etc.) must be considered for the inputs, weights and activation function. And also the precision (number of bits) should be considered. Increasing the precision of the design elements significantly increases the resources used. Accuracy has a great impact in the learning phase; so the precision of the numbers must be as high as possible during training. However during the propagation phase, lower precisions are acceptable. The resulting difference will be small enough to be neglected especially in classification applications.

Many applications require numbers that aren't integers. There are a number of ways that non-integers can be represented. Adding two such numbers can be done with an integer add, whereas multiplication requires some extra shifting. There is various ways to represent the number systems. However, only one non-integer representation has gained widespread use, and that is floating point [1].

The number systems available today on common processors and digital hardware are broadly categorized as floating-point and fixed-point. In a floating-point representation, the total number of bits available is partitioned into an exponent and mantissa [2]. Generally speaking, the mantissa stores the "significant digits" of the value while the exponent scales the significant digits to the desired magnitude. The action of the exponent is to move, or "float," the decimal point depending on the magnitude being represented; thus the term "floating-point." Because floating-point representations are typically at least 32 bits long (IEEE-754 is a popular standard for 32-bit and 64-bit floating-point numbers) [3], there exists simultaneously high precision and high dynamic range. These traits of floating-point numbers allow most algorithms to be ported directly to floating-point implementations with little or no change, and this is the key reason floating-point representations are highly desirable. The disadvantage of floating-point implementations is that they require a significant amount of extra hardware over fixed-point implementations, which translates to higher parts costs, higher power consumption, slower execution, larger chip area, or a combination of these [4].

As the term "fixed-point" implies, fixed-point representations have the binary point at a fixed location. There are two subsets of fixed-point implementations: fractional and integer. In a fractional fixed-point implementation, the binary point is always assumed to be to the left of the most significant

digit. In an integer fixed-point implementation, the binary point is to the right of the least-significant digit. In either case, the arithmetic operations implemented in the hardware are essentially integer, which results in a much simpler arithmetic logic unit in hardware that allows lower cost, lower power consumption, faster execution, smaller chip area, or a combination of these, over that of floating-point implementations.

In this implementation we have used the fractional fixed-point representation to represent the real numbers. Some examples are discussed below for fixed point arithmetic [5], Which we have used in the representation of weights of ANN (which is in real number) and its multiplication and other arithmetic operations done based on the calculation in VHDL programming shown in **Table 8.1**.

Historically, before the floating-point arithmetic was introduced, the only way to develop software calculations was to use fixed-point arithmetic, which is a form of limited precision arithmetic.

- Fixed-point numbers have a fixed number of digits after the decimal point and sometimes also a fixed number of digits before the decimal point.
- This arithmetic is very fast and easy to install.
- However, a programmer has to “handle” the decimal point.

Fixed point is a step between integer math and floating point [5]. This has the advantage of being almost as fast as numeric\_std arithmetic, but able to represent numbers that are less than 1.0. A fixed-point number has an assigned width and an assigned location for the decimal point. As long as the number is big enough to provide enough precision, fixed point is fine for most DSP applications [6]. Because it is based on integer math, it is extremely efficient – as long as the data does not vary too much in magnitude [7].

In Fixed point number system has two new data types are defined as follows:

- 1) “**ufixed**” is the unsigned fixed point,
- 2) “**sfixed**” is the signed fixed point.

type ufixed is array (INTEGER range <>) of STD\_LOGIC;

type sfixed is array (INTEGER range <>) of STD\_LOGIC;

Fixed point arithmetic can be performed using fixed point package available in library ieee\_proposed.

### **Signal a: ufixed (8 downto -23);**

A negative index is used to separate the fraction part of the floating-point number from the exponent. The top bit is the sign bit ('high), the next bits are the exponent ('high-1 downto 0), and the negative bits are the fraction (-1 downto 'low). For a 32-bit representation, that specification makes the number look as follows:

```

0 00000000 000000000000000000000000
| |-----| |-----|
8 7 to 0    ( -1 to -23)
+/-  exp   .   fraction

```

Where the sign is bit 8, the exponent is contained in bits 7–0 (8 bits) with bit 7 being the MSB, and the mantissa is contained in bits –1 -- –23 (32 – 8– 1 = 23 bits) where bit –1 is the MSB. The data widths in the fixed-point package were designed so that there is no possibility of an overflow. This is a departure from the “numeric\_std” model, which simply throws away underflow and overflow bits. According to size of input variables size of output variables can be determined which is shown in **Table 8.1**.

For fixed point: Operation	Result Range
A + B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A - B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A * B	A'left + B'left+1 downto A'right + B'right
A rem B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed /	A'left - B'right+1 downto A'right - B'left
Signed A mod B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed Reciprocal(A)	-A'right downto -A'left-1
Abs(A)	A'left +1 downto A'right
- A	A'left +1 downto A'right
Unsigned /	A'left - B'right downto A'right - B'left -1
Unsigned A mod B	B'left downto Min(A'right, B'right)
Unsigned Reciprocal(A)	-A'right +1 downto -A'left

**Table 8.1: Fixed point Signal size after operation**

### ⌘ For Unsigned numbers

```

signal n1,n2 : ufixed(4 downto -3);
signal n3    : ufixed(5 downto -3);
signal n4    : ufixed(2 downto -4);
signal n5    : ufixed ( 8 downto -6);
signal n6    : ufixed( 9 dwonto -6);
n1 <= to_ufixed (5.75,n1);
n2 <= to_ufixed (6.5,n2);

```

Operands	operations	Coding	Comment
$n1 = 5.75$ $+ n2 = 6.5$ <hr/> $n3 = 12.25$	00101110 $+ 00110100$ <hr/> 01100010	$n3 \leq n1+n2;$	$5.75 = 0\ 0101.110$ (4 downto -3) $6.5 = 0\ 0110.100$ (4 downto -3) $12.25 = 0\ 01100.010$ (5 downto -3)
$n1 = 5.75$ $- n2 = 6.5$ <hr/> $n3 = 0.75$	00101110 $- 00110100$ <hr/> 00000110	$n3 \leq n1-n2;$	$5.75 = 0\ 010.1110$ (4 downto -3) $6.5 = 0\ 011.0100$ (4 downto -3) $0.75 = 0\ 00000.110$ (5 downto -3)
$n1 = 5.75$ $n2 = 6.5$ <hr/> $n6 = 37.375$	00101110 $* 00110100$ <hr/> 0000100101011000	$n3 \leq n1*n2;$	$5.75 = 0\ 010.1110$ (4 downto -3) $6.5 = 0\ 011.0100$ (4 downto -3) $37.375 = 0\ 000100101.011000$ $(9(4+4+1)$ downto $-6(3+3))$
$n1 = 6.75$ $n4 = 1.5$ <hr/> $n5 = 1.5$	00110110 $/ 0011000$ <hr/> 000000100100000	$n3 \leq n1 / n2;$	$6.75 = 00110.110$ (4 downto -3) $1.5 = 0011000$ (2 downto -4) $1.5 = 000000100100000$ $(8(4-(-4))$ downto $-6(-3-2-1))$

Table 8.2: Arithmetic fixed point operation for unsigned numbers

### ⊗ For Signed numbers

signal s1,s2 : sfixed(4 downto -3);

signal s3 : sfixed(5 downto -3);

s1 <= to\_sfixed (5.75,n1);

s2 <= to\_sfixed (6.5,n2);

For signed number if the number is with negative sign then number should be in 2's complement format with sign bit as '1' and all other arithmetic operations are same as unsigned numbers. Size of the answer signals can be determined from the **Table 8.1**. Addition of signed numbers is shown in following

Table 8.3.

Operands	operations	Coding	Comment
$s1 = 5.75$ $s2 = -6.5$ <hr/> $s3 = -0.75$	00101110 $+ 11001100$ <hr/> 111111010	$s3 \leq s1+s2;$	$5.75 = 0\ 0101.110$ (4 downto -3) $6.5 = 1\ 1001.100$ (4 downto -3) $-0.75 = 1\ 11111.010$ (5 downto -3)

Table 8.3: Operation of signed numbers

## 8.1 FPGA Implementation of ANN

There are several options for an electronic system designer to implement any digital logic. These options include discreet logic devices such as Small-Scale Integrated Circuits (SSIC); Programmable Logic Devices (PLDs); Masked-Programmed Gate Arrays (MPGAs) and Field Programmable Gate Arrays (FPGAs). Programming of such a device often involves placing the chip into a special Programming unit, but some chips can also be configured “in-system”. Another name for PLDs is Field Programmable Devices (FPDs). A Programmable Logic Device (PLD) is a set of fully connected macro cells. These macro cells are typically comprised of some amount of combinational logic (for example, AND-OR gates) and a flip-flop. In other words, a small Boolean logic equation can be built within each macro cell .

The Field-Programmable Gate Array (FPGA) has matured over the past years from an in-product personalization of Gate Arrays to a carrier of innovative computing styles. The initial gate array contained a prefabricated collection of transistors and low-level wire segments, that could be personalized by a last series of contact and interconnect fabrication steps. Often support was given from a library of final masks for logic cells. The main purpose was to bring the lumped logic elements of a computing architecture into a single container and thereby save valuable board space [6]. With the advance of microelectronic technology, the capacity of the gate array increased to a level on which it could even contain entire application specific circuits, the ASIC. Still, personalization was performed at the foundry and, though the amount of prefabrication brought some of the cost benefits of mass production, a product series was required to make the concept effective.

Field Programmable Gate Array (FPGAs) are a family of programmable device based on an array of configurable logic blocks (CLBs), which gives a great flexibility in prototyping, designing and development of complex hardware real time systems. Each block is programmed to perform a logic function that can be interconnected, so that the complete logic functions are implemented. The main advantage of FPGA is the flexibility that they afford. ”Field Programmable” means that the FPGA’s function is defined by a user’s program rather than by the manufacturer of the device.

ANNs are biologically inspired and require parallel computations in their nature. Microprocessors and DSPs are not suitable for parallel designs. Designing fully parallel modules can be available by ASICs and VLSIs but it is expensive and time consuming to develop such chips. In addition the design results in an ANN suited only for one target application. FPGAs not only offer parallelism but also flexible designs, savings in cost and design cycle.

The Very high speed integrated circuit Hardware Description Language (VHDL) [8] is heavily used for FPGA programming. VHDL is very powerful (HDL) but very complex syntax language. VHDL

simplifies the development of complex system such as ANNs, because it is possible to model and simulate a digital system from a high level of abstraction with important facilities for modular design. VHDL is a programming language that has been designed and optimized for describing the behavior of digital systems. It has many features appropriate for describing the behavior of electronic components ranging from simple logic gates to complete microprocessors and custom chips. One of the most important applications of VHDL is to capture the performance specification for a circuit, in the form of what is commonly referred to as a test bench.

In this paper an expandable on-chip backpropagation (BP) learning neural network proposed. Large-scale neural networks with arbitrary layers and discretionary neurons per layer can be constructed by combining a certain number of such unit networks. A novel neuron circuit with programmable International Journal of Information Technology and Knowledge Management parameters, which generates not only the sigmoid function but also its derivative, is proposed [9]. The back-propagation (BP) algorithm often provides a practical approach to a wide range of problems. There have been many examples of implementations of general-purpose neural networks with on-chip BP learning.

### **8.1.1 Algorithm for FPGA implementation of ANN based Reactive routing protocol**

The application selected in this work is an AODV Reactive Routing protocol for Mobile Ad-Hoc Network to decide the frequency of Hello Interval to optimize the power [10]. A 2-4-1 feed forward network (two neurons in the input layer, four neurons in the hidden layer and one neuron in the output layer) is implemented on a XILINX Virtex-5 Kit [11]. First the network is trained in MATLAB Neural Networks Processing Toolbox. The output of ANN is given to Qualnet to test the result of Routing protocol in Mobile Ad-hoc Network. Then calculated weights of ANN using MATLAB are written to a VHDL Package file. This file, along with other VHDL coding is compiled, synthesized and implemented with Xilinx ISE software tools. Simulation results are visualized using ISIM and ModelSim. Finally the design is realized on a Xilinx Virtex-5 demo board [11] having the Xilinx FPGA chip. **Figure 8.1** shows the process of implementation of ANN based reactive routing protocol on FPGA Chip. This process is offline and can be make it online by set up the links between software.

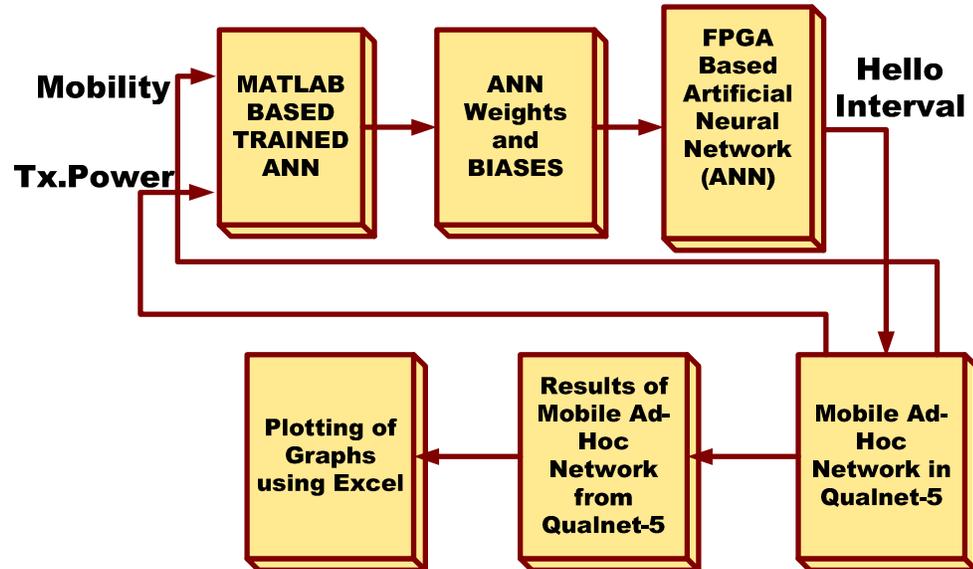


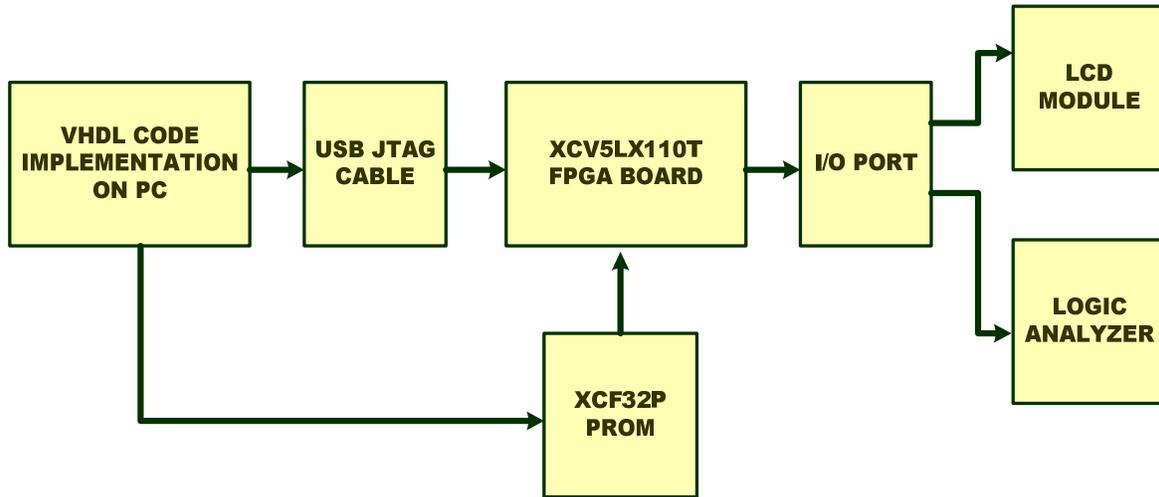
Figure 8.1: Process of FPGA Based ANN for Reactive Routing Protocol for MANET

- **Algorithm for FPGA implementation of ANN based Reactive routing protocol**
  1. Decide the input parameters to the ANN.
  2. Decide the output parameter from the ANN.
  3. Calculate the input and output training pairs of ANN. In this paper it is decided by the Fuzzy Inference System.
  4. Train the ANN using the training pairs. After training, obtain the trained ANN.
  5. Obtain the weights and biases from the Trained ANN and input it to the VHDL code of ANN.
  6. Execute the VHDL code for the decided input, weights and biases for ANN.
  7. Observe the results Using Modelsim OR ISIM.
  8. Repeat the steps 3 to 7 for different inputs and verify the outputs.
  9. Compare the results with MATLAB based ANN.
  10. If the results are satisfactory then load the VHDL code in FPGA Chip.

### 8.1.2 System Block Diagram

In this project work Hello Interval of AODV reactive routing protocol is controlled by ANN for WANET is implemented on FPGA. The process for implementation and execution of VHDL code for FPGA Board is discussed in appendix. For the implementation of VHDL code ISE Design Suite 13.1 [13] was used. For the loading of VHDL code in FPGA chip was done using iMPACT. The Hardware Implementation includes the FPGA Evaluation board XCV5LX110T, Logic Analyzer, I/O port, LCD

Module, USB JTAG Cable. The System Block Diagram of FPGA Implementation of ANN based HI is shown in **Figure 8.2**.



**Figure 8.2: System Block Diagram of FPGA Implementation of ANN AODV**

In this VHDL code is loaded to the FPGA Board using USB JTAG cable from the PROM. PROM is used to store the program from which FPGA board can execute the program. External asynchronous memories and memory mapped devices can be added to the FPGA Board, including nonvolatile memory that can be used to boot the DSK upon reset. The output is given to General Purpose Input Output Port (GPIO) and the Output devices like LCD and Logic analyzer can be interfaced with the GPIO to analyze the output.

### 8.1.3 System Software simulation Results

Simulation of AODV routing protocol with ANN were carried out in Qualnet simulator version 5 on the Mobile Ad-Hoc network with the 50 nodes [10]. The MATLAB was used to design ANN. The hardware implementation of ANN is done on FPGA kit with the programming language of VHDL. The Simulation results visualized using ISIM and compared with the MATLAB implementation.

The Device utilization summary for the Purelin type ANN when the output of ANN observed on GPIO pins using logic analyzer is shown in **Table 8.4** while for Tansig type ANN is shown in **Table 8.5**.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	637	69120	0%
Number of fully used LUT-FF pairs	0	637	0%
Number of bonded IOBs	45	640	7%
Number of DSP48Es	16	64	25%

Table 8.4: Device Utilization Summary of Purelin ANN from Xilinx ISE 13.1

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	1188	69120	1%
Number of fully used LUT-FF pairs	0	1188	0%
Number of bonded IOBs	45	640	7%
Number of DSP48Es	16	64	25%

Table 8.5: Device Utilization Summary of Tansig ANN from Xilinx ISE 13.1

Figure 8.3 (a) shows the output of Purelin ANN model observed from the ISIM tool after simulating the program for XILINX Virtex-5 Kit [11]. In the simulation output waveforms it shows 14 bit inputs (Transmission Power and Mobility of nodes) viz. p1\_1 and p2\_1 and the 14 bit output (Hello Interval) viz. l3purelin\_out.

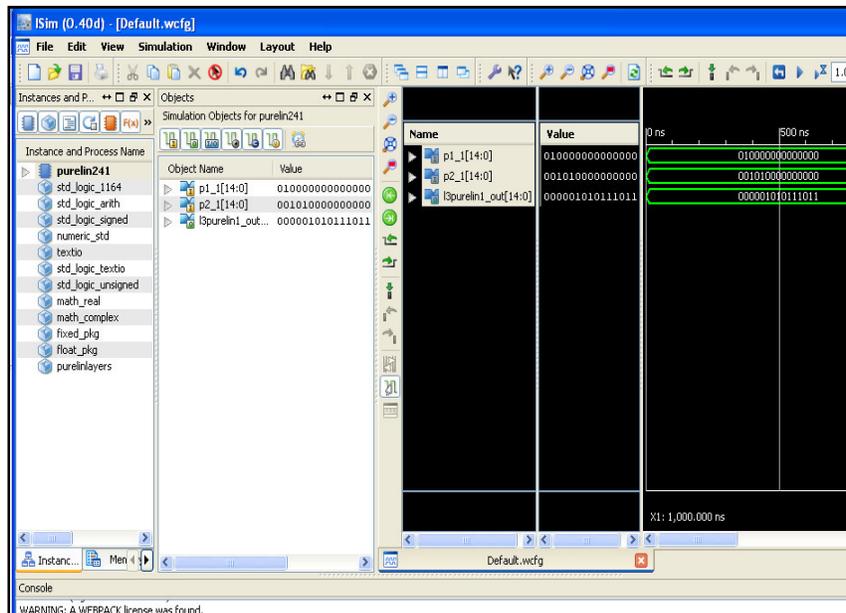
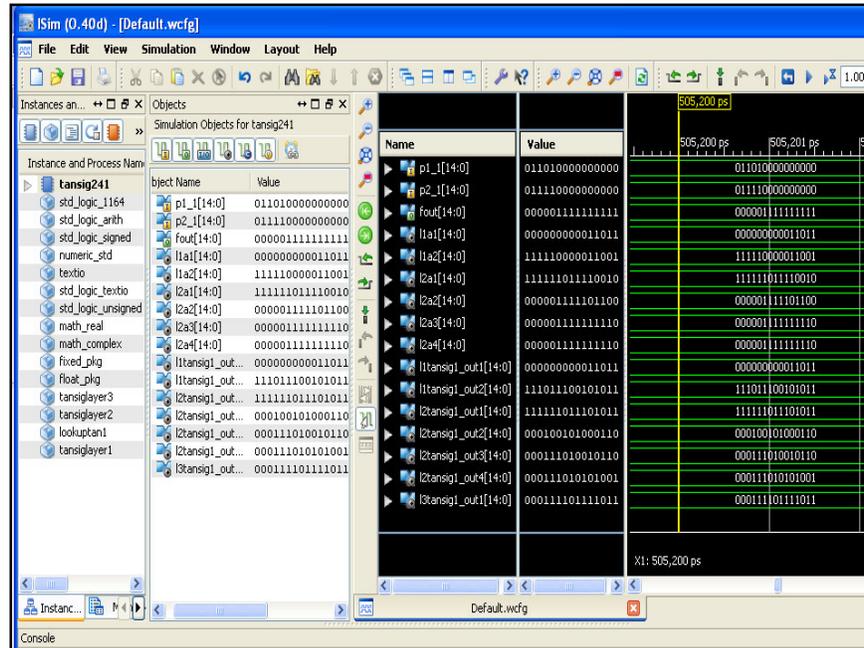


Figure 8.3(a): Output of Purelin ANN from ISIM Using VHDL code



**Figure 8.3(b): Output of Tansig ANN from ISIM Using VHDL code**

**Figure 8.3(b)** shows the output of Tansig ANN model input and output waveforms observed in the ISIM tool after simulating the program for XILINX Virtex-5 Kit. The inputs (Transmission Power and mobility of nodes) viz. p1\_1 and P2\_1 of 14 bit and the output viz. Hello Interval decided by ANN as fout signal of 14 bit can be visualized from it.

ANN model has the two inputs one is Tx power and other input is mobility and the output is hello interval. The output is tested with the input value of Tx power in the range of 8mW to 14 mW and mobility is in the range of 5 m/s to 15 m/s for WANET. The result of both types viz. Purelin and Tansig ANN using MATLAB and VHDL code implemented is verified by calculating relative error.

Implementation of Tansig and Purelin type ANN was implemented on MATLAB and then it is realized on Xilinx ISE13.1 using VHDL code. This VHDL code then loaded on FPGA Virtex -5 Evaluation Platform. The relative difference between Hardware and Software Implementation of ANN was compared in the terms of relative difference. The relative difference of MATLAB and VHDL result for corresponding mobility and transmission power of the nodes in WANET in terms of Bar chart is shown in **Figure 8.4 (a)** and **(b)** for Purelin and Tansig type ANN respectively.

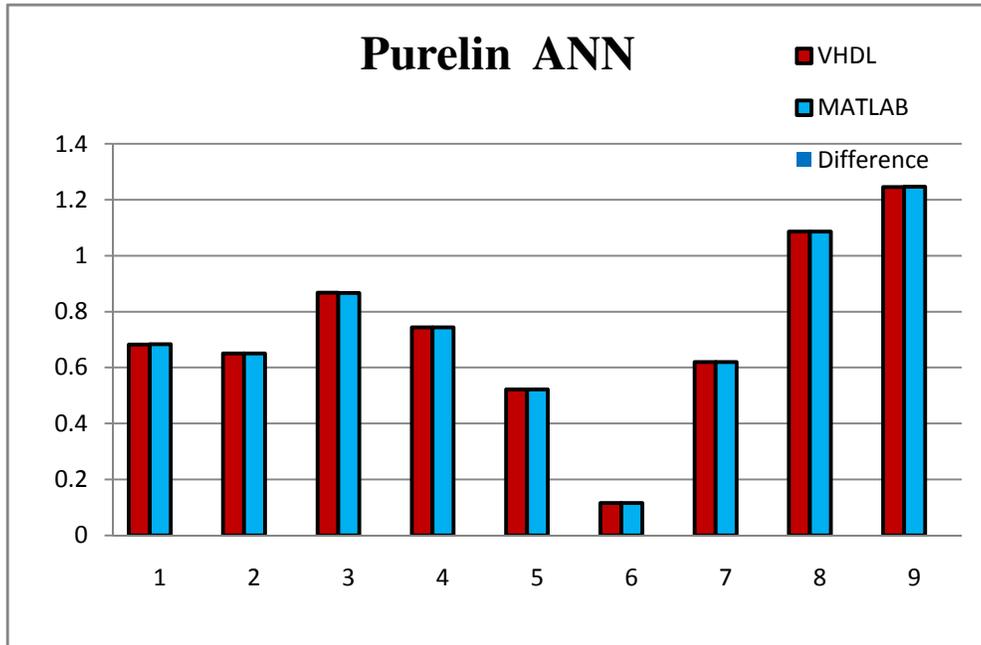


Figure 8.4 (a): Relative difference of Purelin type ANN

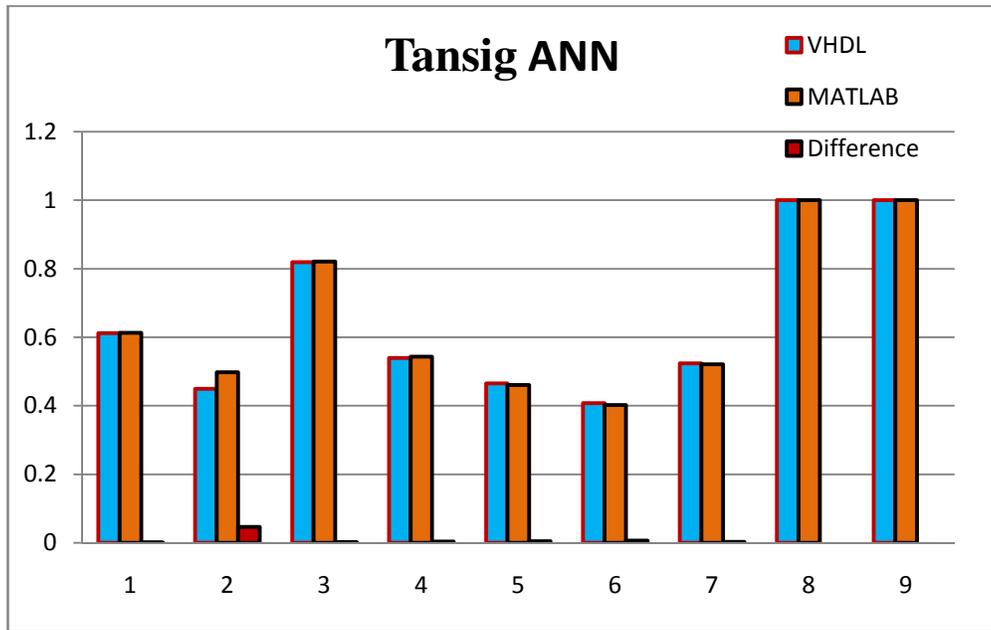


Figure 8.4(b): Relative difference of Tansig type ANN

Table 8.6 shows the results comparisons of the implementation of Purelin and Tansig type ANN in MATLAB and VHDL in terms of the relative difference between both the implementations.

Input Parameter		Purelin ANN Based Hello Interval		Relative Difference	Tansig ANN Based Hello Interval		Relative Difference
Txpower	Mobility	USING MATLAB	USING FPGA		USING MATLAB	USING FPGA	
8	5	0.6833	0.6828	0.0006	0.6130	0.6124	0.0006
9	8	0.6505	0.6504	0.0001	0.4978	0.4497	0.0461
12	12	0.8671	0.8672	0.0001	0.8206	0.8191	0.0015
10	9	0.7437	0.7435	0.0002	0.5430	0.5395	0.0035
11	15	0.5217	0.5224	0.0007	0.4610	0.4652	0.0042
8	14	0.1159	0.1164	0.0005	0.4020	0.4082	0.0062
8	6	0.6202	0.6203	0.0001	0.5213	0.5241	0.0028
13	11	1.0860	1.0859	0.0001	0.9999	0.9999	0.0000
12	06	1.2460	1.2451	0.0009	1.0000	1.0000	0.0000

Table 8.6: Comparison of results of MATLAB and FPGA Implementation of Purelin ANN

### 8.1.4 Hardware Setup and Implementation

The hardware setup of Implementation of Tansig and Purelin type ANN on XCV5LX110T FPGA kit with the Xilinx ISE 13.1 is shown in Figure 8.5(a), 8.5(b) and 8.5(c).



Figure 8.5(a): Hardware setup of FPGA Implementation of ANN based HI

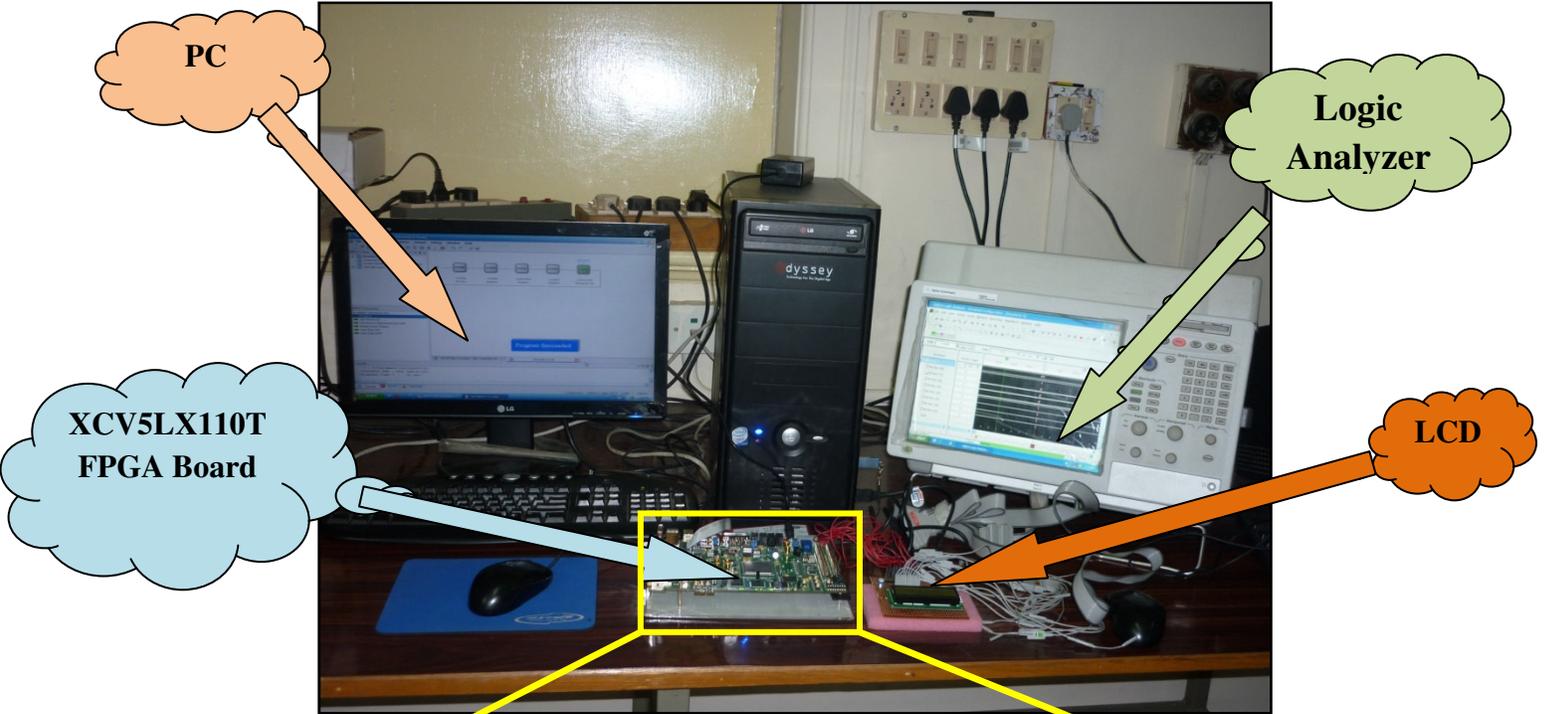


Figure 8.5(b): Hardware Setup of FPGA implementation of ANN

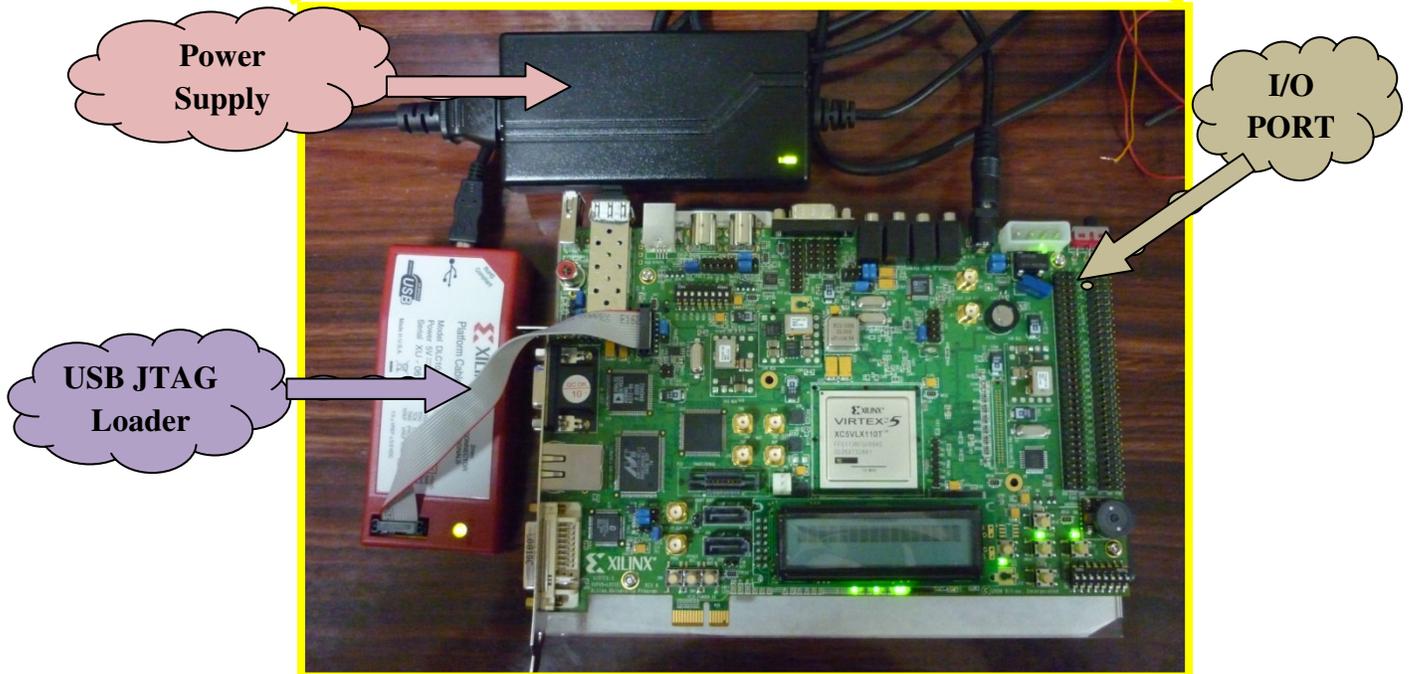
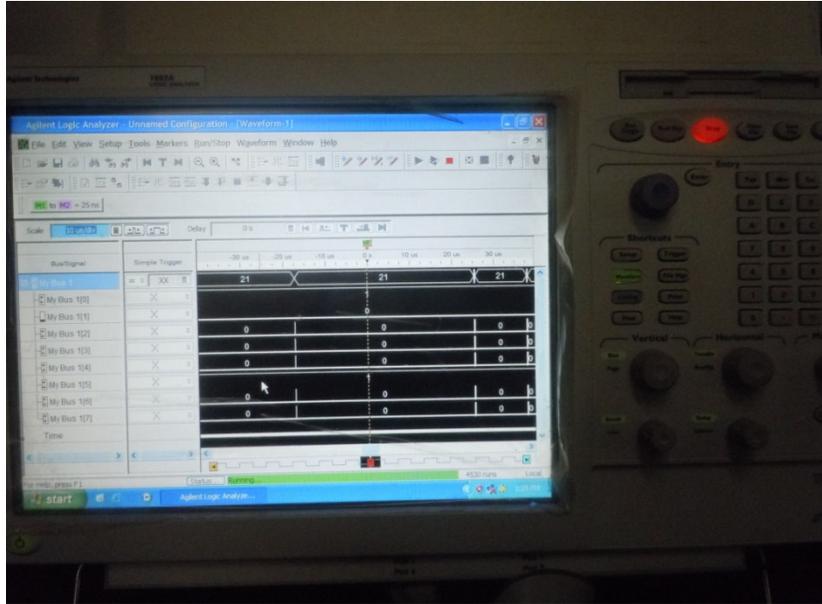


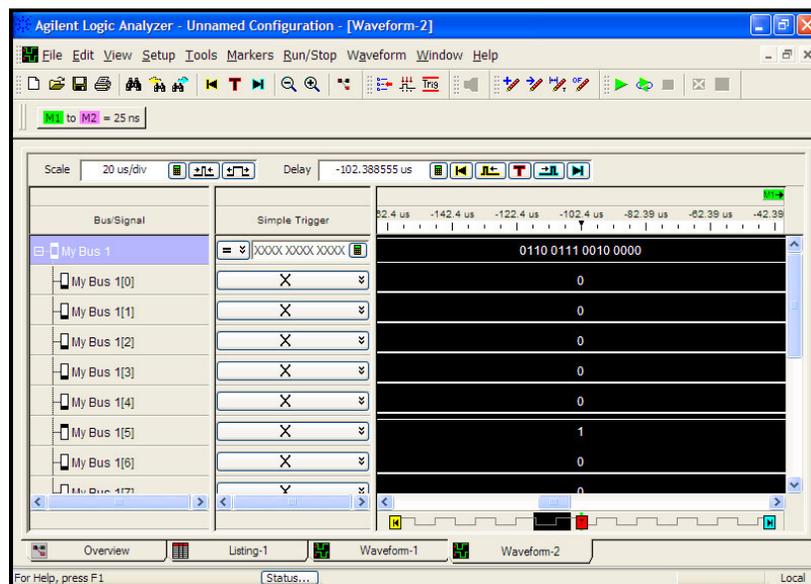
Figure 8.5(c): Enlarge View of FPGA Board in Hardware Setup

The Outputs were kept on GPIO pins of FPGA Kit and observed on Logic analyzer is shown in **Figure 8.6(a)** for Purelin and in **Figure 8.6(b)** for Tansig type ANN.



**Figure 8.6(a): Snap Shot of Output of Purelin ANN on Logic Analyzer**

The 14 bit output of ANN was given to the GPIO pins and logic analyzer was interfaced to visualize the status of each pin. In the Logic analyzer Pod 1 was used of 16 bit with the GPIO pins. My bus of 16 bit with the bus bit 0 indicates LSB and bus bit 14 indicates MSB of the observed output signal.



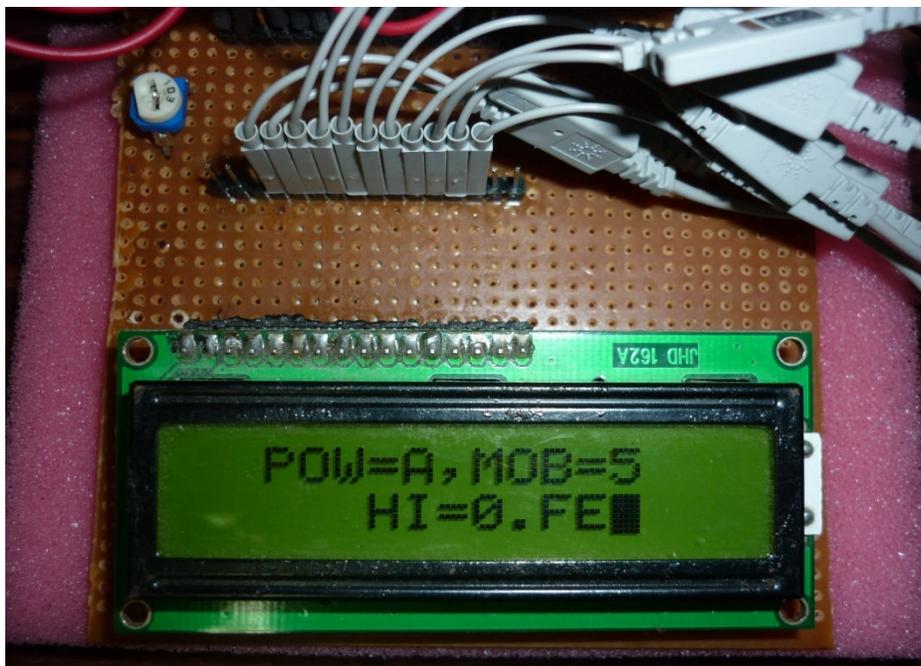
**Figure 8.6(b): Snap Shot of the output of Tansig ANN observed on GPIO pins using Logic Analyzer**

The results displayed on LCD with the information of inputs and output of both the type of ANN viz. Purelin and Tansig. The inputs are Transmission power (POW) measured in mW and Mobility (MOB) (m/s) of nodes and output of ANN as Hello Interval (HI) measured in seconds shown in **Figure 8.7**.



**Figure 8.7: Hardware Setup of FPGA implementation of ANN with LCD**

Output of Tansig ANN with the transmission power of 10 mW (0A h) and mobility of the nodes with 5 m/s (05h), output as Hello Interval (HI) of 0.994 sec (0.FEh) decided by ANN displayed on LCD after loading the program in FPGA kit is shown in **Figure 8.8(a)**.



**Figure 8.8(a): Snap Shot of Tansig type ANN Input and Outputs Are Displayed on LCD**

Output of Purelin ANN with the transmission power of 10 mW (0A h) and mobility of the nodes with 5 m/s (05h), output as Hello Interval (HI) of 0.994 sec (0.FEh) decided by ANN displayed on LCD after loading the program in FPGA kit is shown in **Figure 8.8(b)**.

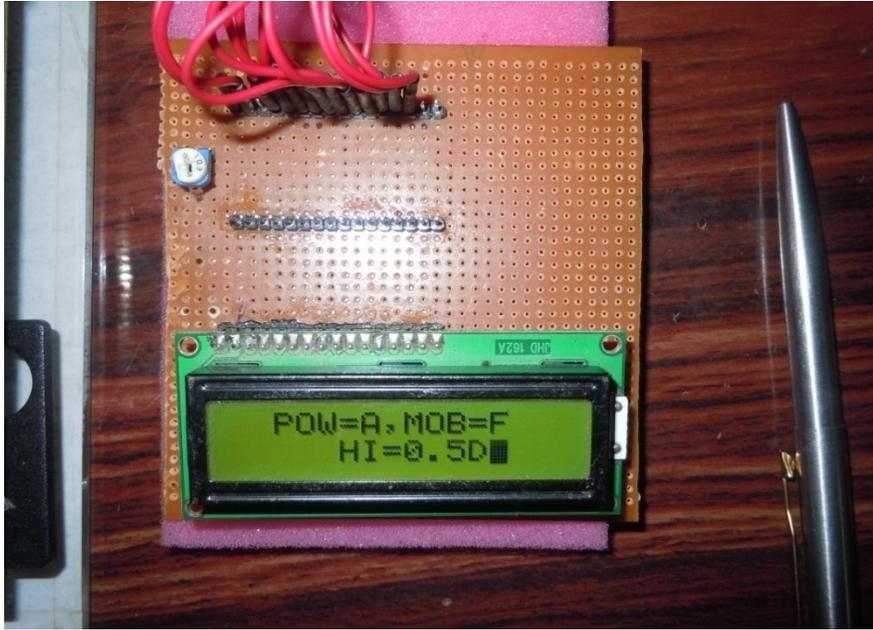


Figure 8.8(b): Snap Shot of Tansig type ANN Input and Outputs Are Displayed on LCD

## 8.2 Files Used for FPGA Implementation of ANN

Table 8.7 is a list of VHDL files and VHDL packages used to Implement Purelin and Tansig type ANN in Xilinx ISE 13.1( xxx241.\* = purelin/tansig ), while Table 8.8 gives list of Libraries used for both implementation.

Sr.No.	Name	Purpose
1.	xxx241.xise	Xilinx ISE 13.1 project file
2.	xxx241.vhd	Vhdl file to write the vhdl code
3.	xxxlayers.vhd	Vhdl package for implementing layers of ANN
4.	xxx241_pin.ucf	Assignment of hardware pins of FPGA board to the input and output variables.
5.	xxx241.mcs	PROM files are PROM programming files generated by iMPACT using the PROM Formatter tab in the File Generation mode. They are ASCII text files used to specify configuration data.
6.	xxx241.bit	Bit files are Xilinx FPGA configuration files generated by the Xilinx FPGA design software. They are proprietary format binary files containing configuration information.
7.	xxx.ncd	Timing Analyzer opens with the timing report which

		was created during the Generate Post-Map Static Timing process and the Post-Map NCD loaded.
8.	xxx.ngc	A binary Xilinx implementation netlist. The logic implementation of certain CORE Generator IP is described by a combination of a top level EDN or NGC and possibly one or more lower level NGC files.
9.	xxx241.ngd	Native generic database file. This is a translated netlist file that describes the logical design reduced to Xilinx® primitives. It is sometimes called the design file, and serves as input to the mapper.
10.	xxx241.ngr	Timing Analyzer opens with the timing report which was created during the Generate Post-Map Static Timing process and the Post-Map NCD loaded.
11.	xxx241.par	When you run the Place and Route (PAR) process, this process is run automatically and a Post-Place and Route Static Timing Report is automatically generated.
12.	xxx241functions16novxcf32p.iaf	An iMPACT Archive file is generated by the Save Archive menu command. This command collects all configurations and configuration related files together and creates a context independent directory structure containing all these files.
13.	xxx41.bld	After running the Implement Design process, you can analyze the reports generated during the underlying Translate, Map, and Place and Route processes to find out if we need to improve design performance prior to programming and configuring your device. The Translation Report (.bld extension), Map Report (.mrp extension), and Place and Route Report (.par extension) are output to your project directory, and we can view these reports in the Design Summary.
14.	d12.cfi	A Configuration Format Information file is created by PROMGen and used for 8, 16, and 32 MB PROMs.
<b>Table 8.7: VHDL files related to FPGA Implementation of ANN</b>		

Library Name	Files
ieee	std_logic_1164
	std_logic_arith
	std_logic_signed
	numeric_std
	math_real
	math_complex
ieee_proposed	fixed_pkg
	float_pkg

**Table 8.8: libraries used in FPGA Implementation of ANN**

### 8.3 Neuro-Fuzzy AODV

The acronym ANFIS derives its name from Adaptive Neuro-Fuzzy Inference System. Using a given input/output data set, the toolbox function ANFIS constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm alone or in combination with a least squares type of method to facilitate the fuzzy systems to learn from the data. ANFIS maps input through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs, can be used to interpret the input/output map.

The parameters associated with the membership functions changes through the learning process. The computation of these parameters (or their adjustment) is facilitated by a gradient vector. This gradient vector provides a measure of how well the fuzzy inference system is modeling the input/output data for a given set of parameters. When the gradient vector is obtained, any of several optimization routines can be applied in order to adjust the parameters to reduce some error measure. This error measure is usually defined by the sum of the squared difference between actual and desired outputs. ANFIS uses either back propagation or a combination of least squares estimation and backpropagation for membership function parameter estimation [13].

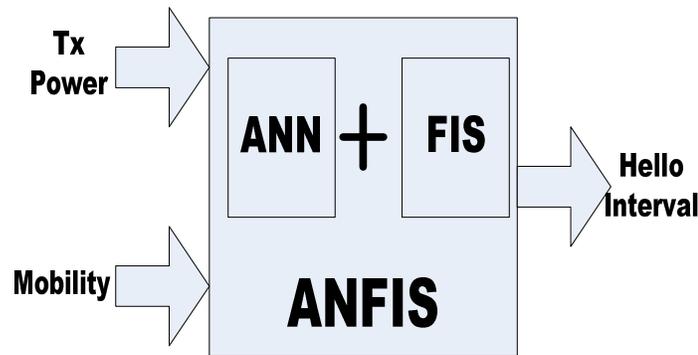
#### 8.3.1 ANFIS Based AODV<sup>1</sup>

The transmission power, strength by which signal is sent, is a parameter that determines the number of neighbours for nodes in MANET.

<sup>1</sup> **Published Paper in 4th IEEE INTERNATIONAL CONFERENCE** proceeding “**Performance Evaluation of Reactive Routing Protocol using Parametric Decision based on ANFIS for MANET using Qualnet**” in **International Journal of Computing Science & Communication Technologies (IJCSCT, ISSN-0974-3375) of International Conference On Advance Computing And Communication Technologies ICACCT- 2010**”sponsored by IEEE Delhi Section, IEEE Computer Society Chapter, Delhi Section & IETE Delhi Centre on 30<sup>th</sup> October 2010.

If it is too low, signal will reach a few neighbours only and its links with those neighbours are easy to break. Hello interval must be small enough to get a fast update for neighbourhood changes. In contrast high transmission power leads to a high average number of its neighbours and increases the lifetime of its links. Hello interval must be long due to fewer changes in the node's neighbourhood. The high speed of a node in MANET results in a high probability of losing some of the current neighbours and acquiring new ones. As the nodes moves fast their links lifetime with their neighbours is small. Hello Interval time is small to send more Hello messages to check the expected links breaks.

The potential benefits of Adaptive Neuro-Fuzzy Inference System (ANFIS) [14] extend beyond the high computation rates provided by massive parallelism. The neural network models are specified by the net topology, node characteristics, and training or learning rules. These rules specify an initial set of weights and indicate how weights should be adapted during use to improve performance. Inputs of the ANFIS are Transmission power and Mobility of the node (speed) and the output of the system is hello interval which is shown in the **Figure 8.9**. The neural Network is trained to decide the hello interval based on information transmission power and speed/mobility of nodes.



**Figure 8.9: Block Schematic of ANFIS**

### 8.3.2 ANFIS Configuration

Since it is possible to generate I/O training pairs from the existing environment and classical procedures, we have selected a feed forward ANN with multi layers perceptron model, with 2 nodes in the input Layer, 10 hidden layers and 1 node in the output layer. The input layer neurons have linear activation characteristics while the hidden and output layers have a hyperbolic tan-type activation function to produce bipolar outputs. C module has been developed to generate I/O pairs (P, T) for training ANFIS. P indicates input and T indicates target output. Conventional Back propagation training method is used. The MATLAB [15] is used to load the training pairs decided by the Sugeno Fuzzy Inference System (FIS) to train the neural network using ANFIS Editor:

ANFIS is a network-type structure similar to that of a neural network, which maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs, can be used to interpret the input/output map. The parameters

associated with the membership functions will change through the learning process. The computation of these parameters (or their adjustment) is facilitated by a gradient vector, which provides a measure of how well the fuzzy inference system is modelling the input/output data for a given set of parameters [14].

Once the gradient vector is obtained, any of several optimization routines could be applied in order to adjust the parameters so as to reduce some error measure (usually defined by the sum of the squared difference between actual and desired outputs). ANFIS uses either back propagation or a combination of least squares estimation and gradient method for membership function parameter estimation. The internal architecture of ANFIS is shown in Figure 8.10.

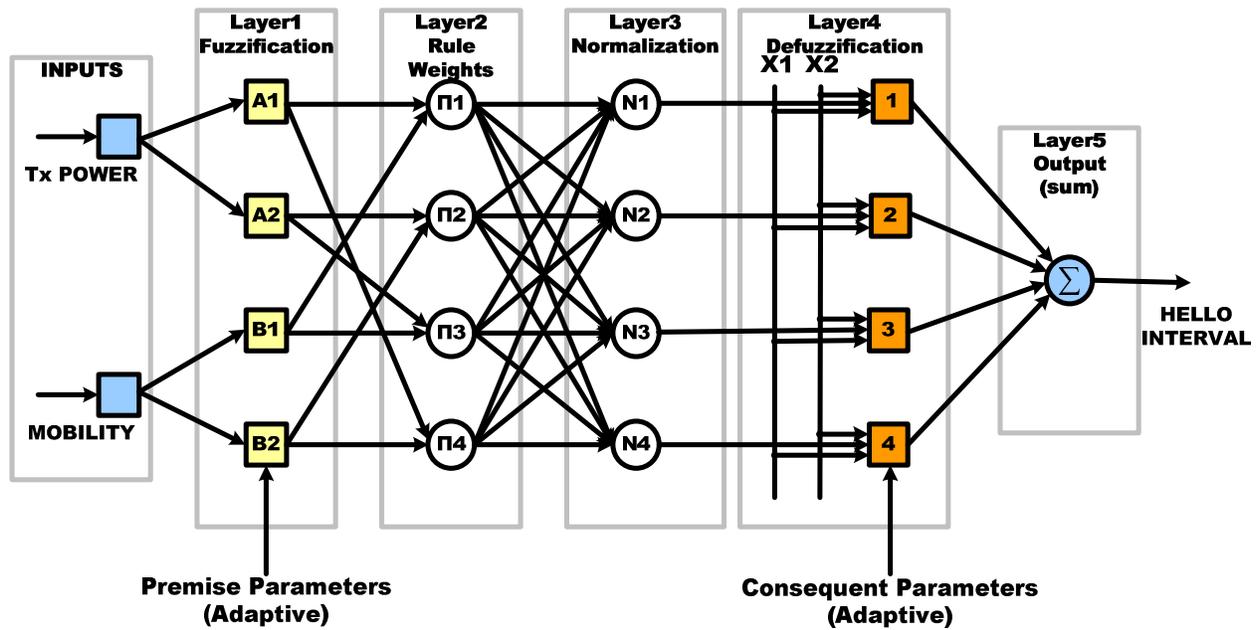


Figure 8.10: Internal architecture of ANFIS

Figure 8.11 shows the training pairs loaded to train the neural network using ANFIS editor in MATLAB.

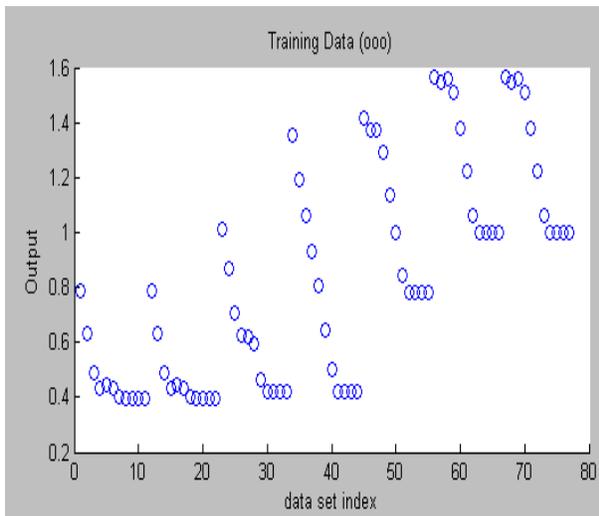


Figure 8.11: Training data loaded for ANFIS

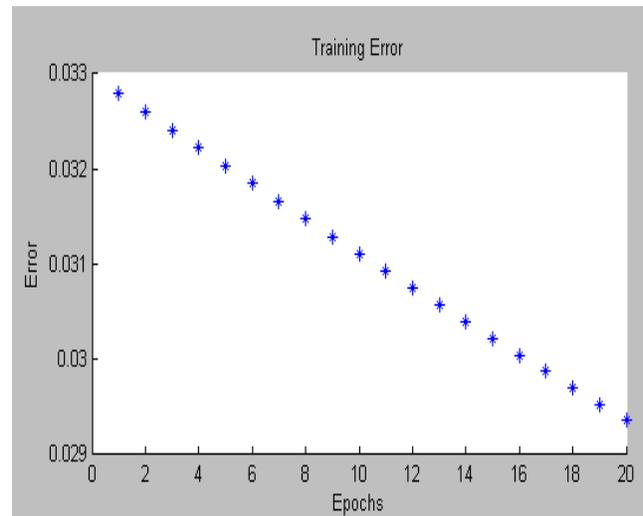
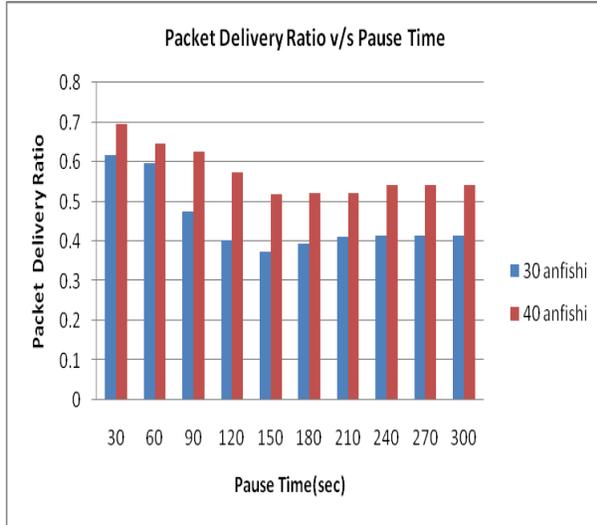


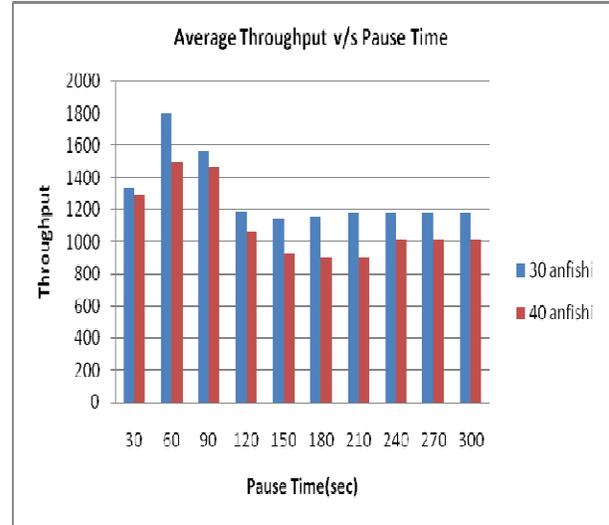
Figure 8.12: Training Error of ANFIS for Hello

Interval (HI)

Training of ANFIS tries to minimize the error within 20 epochs. **Figure 8.12** shows the training error of the ANFIS network.

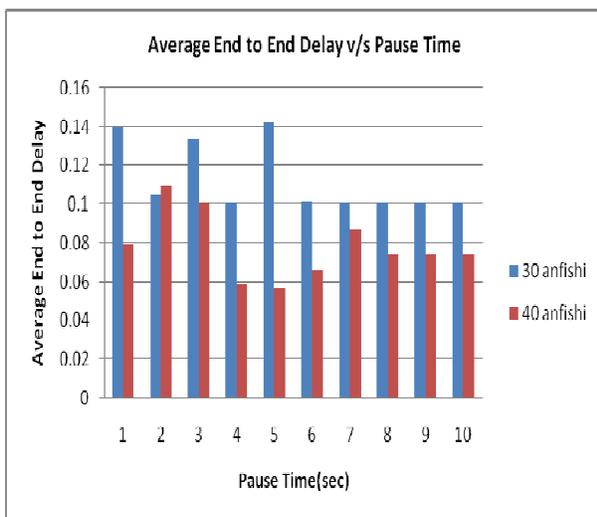


**Figure 8.13: Average Packet Delivery Ratio V/S Pause time for nodes 30, 40**

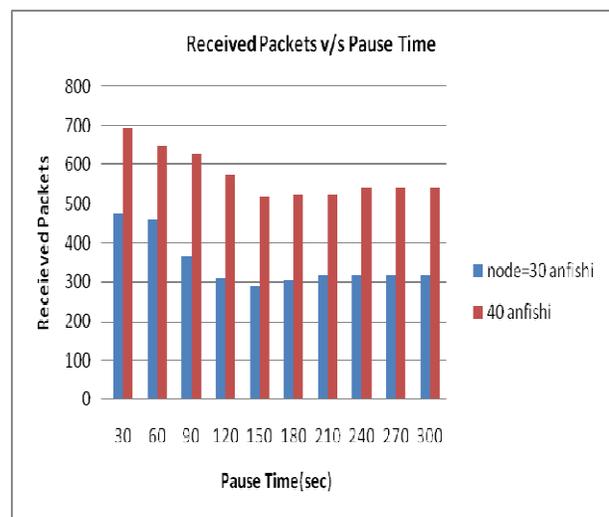


**Figure 8.14: Average Throughput V/S Pause Time for nodes 30, 40**

**Figure.8.13** shows the Packet Delivery Ratio and **Figure 8.14** shows Average Throughput of the AODV protocol for the number of nodes 30 and 40. The Packet Delivery Performance for the hello interval determined using ANFIS is better for higher number of nodes. The Throughput and Received packets performance for the ANFIS Hello interval is also better for the higher node density in the comparison of the nodes 30 and 40.



**Figure 8.15: Average End to End Delay V/S Pause Time for nodes 30, 40**



**Figure 8.16: Average Received Packets V/S Pause Time for nodes 30, 40**

Here the Average end to end delay is higher in the case of nodes=30 while it is less in nodes=40, which can be observed from the **Figure.8.15**. Average Received Packets for the node density of 30 and 40 is shown in **Figure 8.16**.



**Figure 8.17: Average comparison of performance metrics for nodes 30 and 40.**

**Figure 8.17** shows the performance of the proposed method using pie chart for the nodes 30 and 40 and we can observe from that for the higher density of nodes proposed method performs better. From the figure we can also observe that the packet delivery ration and the number of received packets for the node density 40 is higher than the node density 30, also the throughput of the node density 40 is equivalent to the node density 30. The Average End to end delay for higher density node is less.

## Summary

In this chapter FPGA implementation of ANN based Hello Interval parameter of reactive routing protocol AODV has been described. Feed forward type Multilayer Perceptron (MLP) neural network of the Purelin and Tansig type is used to decide interval between hello messages instead of using fixed interval of 1 sec or 1.5 sec. Result comparison of the FPGA implementation of Purelin and Tansig type ANN with the Matlab implementation done by calculating relative error. Also the same Hello interval parameter for reactive routing protocol AODV is decided by ANFIS. The results of ANFIS based Hello Interval is compared with the traditional fixed values of Hello Interval of 1sec and 1.5 sec.