Chapter 8

Embedded Hardware : WSN

This chapter divides in two sections A and B. A section implements trained Artificial Neural Network (ANN) in MATLAB using VHDL programming language and then realized on FPGA kit. B section describes the real time test bed hardware implementation of Wireless Sensor Network (WSN).

SECTION A: Configuration of ANN on FPGA Kit

8.1 Short introduction to FPGAs

Field Programmable Gate Arrays (FPGAs) made their appearance in 1985 when Xilinx started to manufacture the XC2064 [1]. The general architecture of an FPGA structure is composed of four basic reconfigurable elements: Programmable Logic Blocks (PLBs) which is the most significant part that provides physical support for the program downloaded on FPGA, embedded memory, programmable I/O cells which provides input and output for FPGA and makes it possible to communicate outside the FPGA, and programmable interconnections (Programmable Interconnect (PI)) which connects the different part of FPGA and allows them to communicate with each other [2,3]. The way in which these elements are distributed inside the device defines the technical characteristics of each FPGA family. These structures consist of routing channels and programmable switches. Routing process is effectively connection logic blocks exist different distance the others [4].

FPGAs can be programmed via interfaces based on Hardware Description Languages (HDL); the most popular one is the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language, commonly known as VHDL. The process to design an application with an FPGA consist of six main phases: 1) definition of the initial requirements; 2) choice of the appropriate device; 3) writing of the VHDL code; 4) synthesis to map the application onto the resources of the FPGA; 5) simulation; 6) programming of the FPGA (if the simulation succeeds).

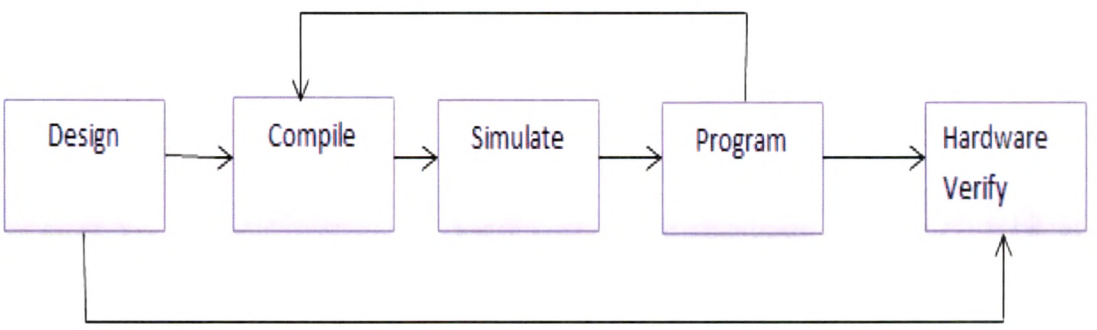

Figure 8.1: Design Flow

The flow in the FPGA hardware is shown in Figure 8.1.

8.2 FPGA design implementation of ANN

FPGAs are chosen for implementation ANNs with the following reason:

- They can be applied a wide range of logic gates starting with tens of thousands up to few millions gates.
- They can be reconfigured to change logic function while resident in the system.
- FPGAs have short design cycle that leads to fairly inexpensive logic design.
- FPGAs have parallelism in their nature. Thus, they have parallel computing environment and allows logic cycle design to work parallel.
- They have powerful design, programming and syntheses tools.

Artificial neural network based on FPGAs has fairly achieved with classification application. The programmability of reconfigurable FPGAs yields the availability of fast special purpose hardware for wide applications.

There are two problems during the hardware implementation of ANNs. How to balance between the need of reasonable precision (number of bit), that is important for ANN and the cost of more logic area associated with increased precision. How to choose a suitable number format that dynamic range is large enough to guarantee that saturation will not occur for a general-purpose application. So before beginning ANN's based FPGAs system design with VHDL, number format (floating point, fixed point etc.) and precision which used for inputs, weighs and activation function must be considered. This important that precision of the numbers must be as high as possible, are used during training phase. This is because precision has a great impact in the learning phase [5]. However, low precision is used during the propagation phase [6]. So especially in classification's applications the resulting errors will be small enough to be neglected [6, 7, 8]. Floating point offers the greatest amount of dynamic range, making it suitable for any application so it would be the ideal number format to use. In this implementation we have used the fractional fixed-point representation to represent the real numbers.

A 2-10-1 feed forward network (two neurons in the input layer, ten neurons in the hidden layer and one neuron in the output layer) is implemented on a Xilinx Spartan-6 LX45 FPGA demo board (features are covered in appendix)[9]. The design implementation is shown in Figure 8.2. It shows the total design flow using MATLAB and Xilinx. The MATLAB program consists of the built and learning programs of ANN.

After the leaning procedure, weights data are fixed and saved to a file. Then transmit the weights to the Xilinx.

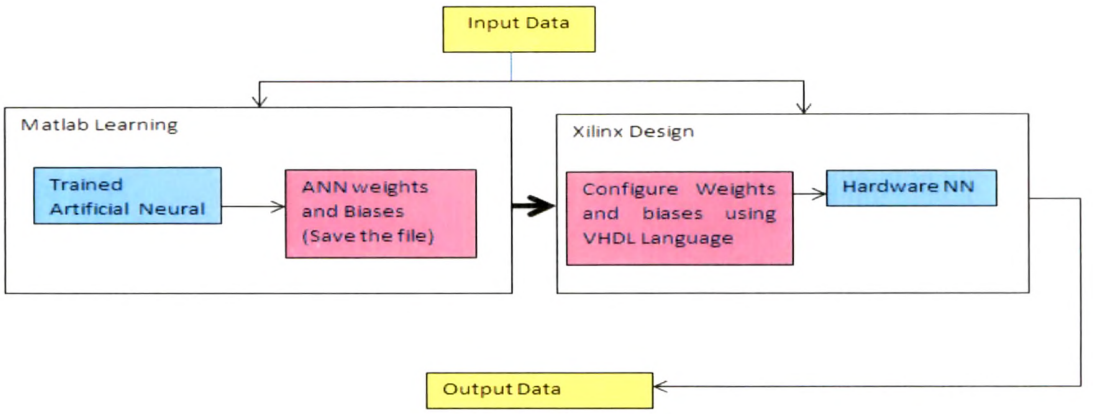

Figure 8.2: Design implementation of ANN on FPGA

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 Spartan 6 XC6SLX45 [9].

8.3 Design Implementation and Simulation Results

Design Algorithm of ANN implementation on FPGA Spartan 6 is described as follow:

Design Algorithm:

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.
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.

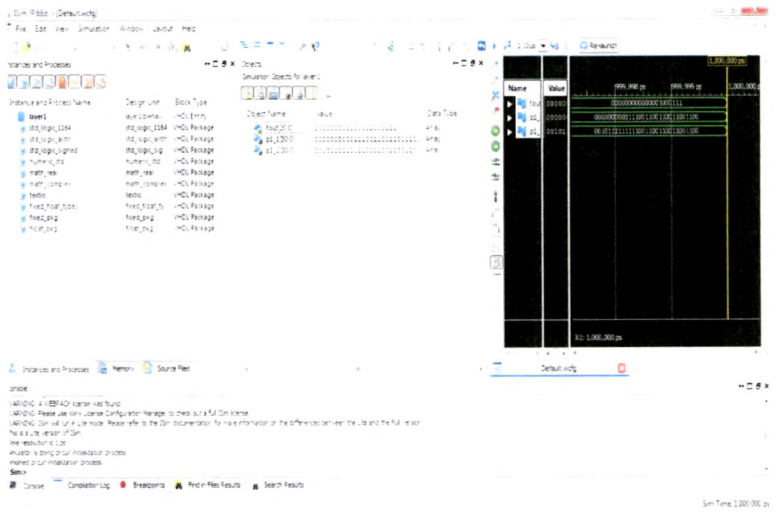

Figure 8.3: Output of ANN from ISIM Using VHDL code

Figure 8.3 shows the output of ANN model observed from the ISIM tool after simulating the FPGA program for Spartan 6 XC6SLX45 demo kit. Output shows waveforms of inputs p_1 and p_2 where two inputs are data rate and interarrival time deciding packet size by ANN model (discussed in chapter 7) and final output configure on FPGA.

The results of ANN using MATLAB and implemented in VHDL code are verified by calculating relative error**.** The following Table 8.1 compares the output of the complete neural network calculated using MATLAB program with ANN implemented using FPGA technique. The relative difference between Hardware and Software Implementation of ANN was compared in the terms of relative difference is shown in Figure 8.4.

Input Parameters		**Packet Size Using Calculation**	**ANN based Packet size**		**Relative Difference (using MATLAB and using FPGA)**
Interarrival Time (ms)	**Data Rate (bits/sec)**		**Calculated using MATLAB**	**Designed using FPGA**	
1.8	82	150	181	175.78	5.22
	164.473	300	322	312.5	9.5
	274.122	500	509	507.81	1.19
	383.771	700	695	693.35	1.6499
	493.421	900	880	878.91	1.09

Table 8.1 Comparison of ANN results of MATLAB and FPGA

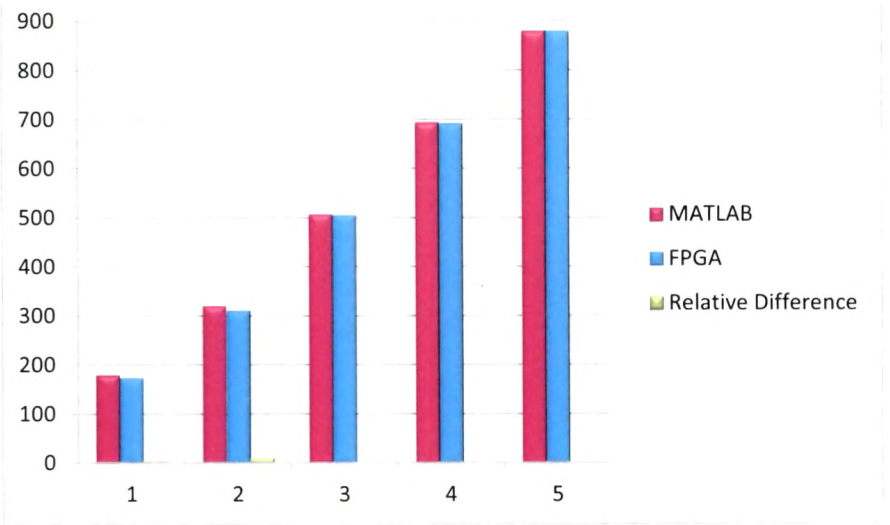

Figure 8.4: Relative Difference

8.4 Test-bed Hardware Implementation

The real time hardware implementation of ANN configuration on Spartan 6 kit setup is shown in below Figure 8.5 and 8.6.

The plus point of this Digilent Atlys Spartan 6 kit is easy way of programming the chip. The kit has Adapt system providing simplified programming interface and many additional features as described in the appendix. The Adept port is compatible with Xilinx's iMPACT programming software if the Digilent Plug-In for Xilinx Tools is installed on the host PC (download from the Digilent website's software section). Following Figure 8.7 shows the snap shot of programming a bit file on the Spartan 6 kit with Digilent Adept Software. Figure 8.8 shows the output on LED for given combination of the inputs Data Rate and Inter Arrival Time (p_1 and p_2).

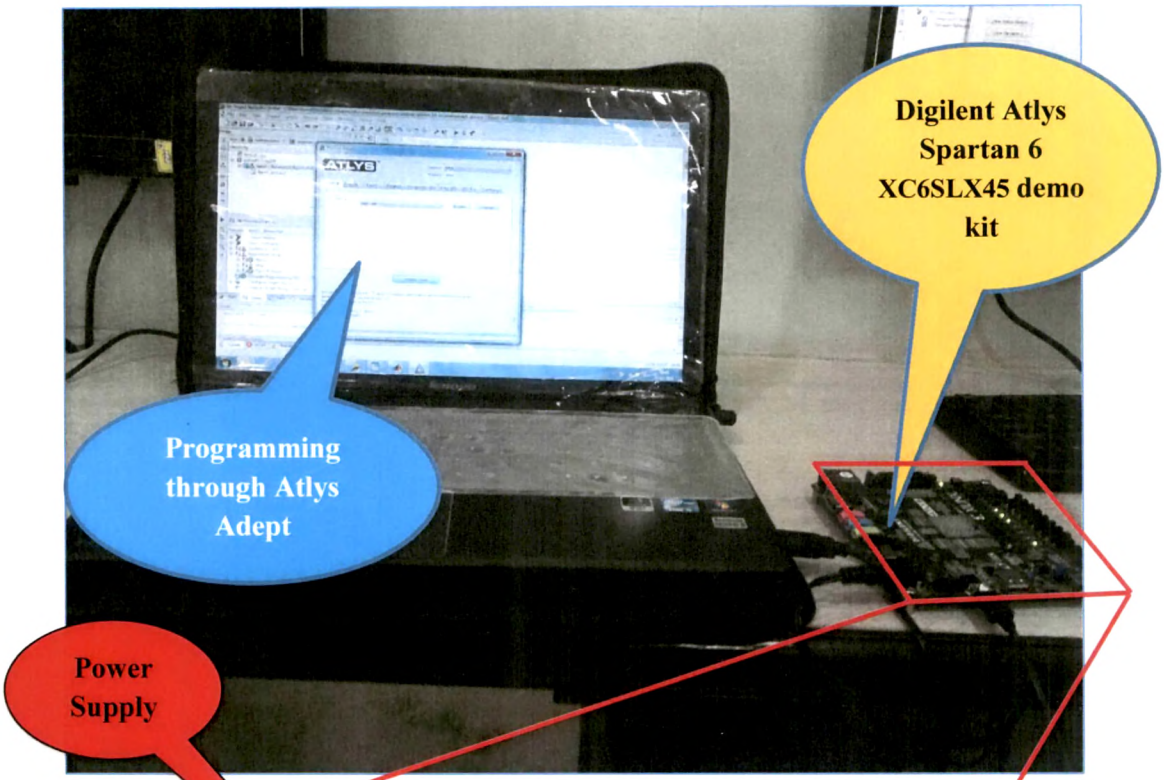

Figure 8.5 Hardware setup of ANN configuration on FPGA kit

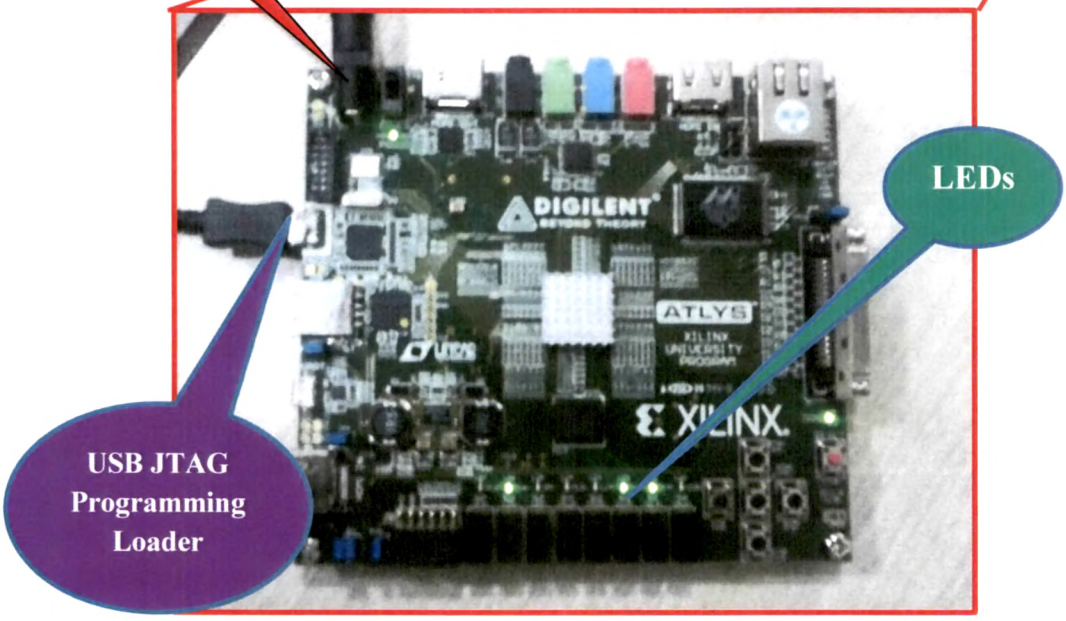

Figure 8.6: Enlarge view of Spartan 6 demo Kit

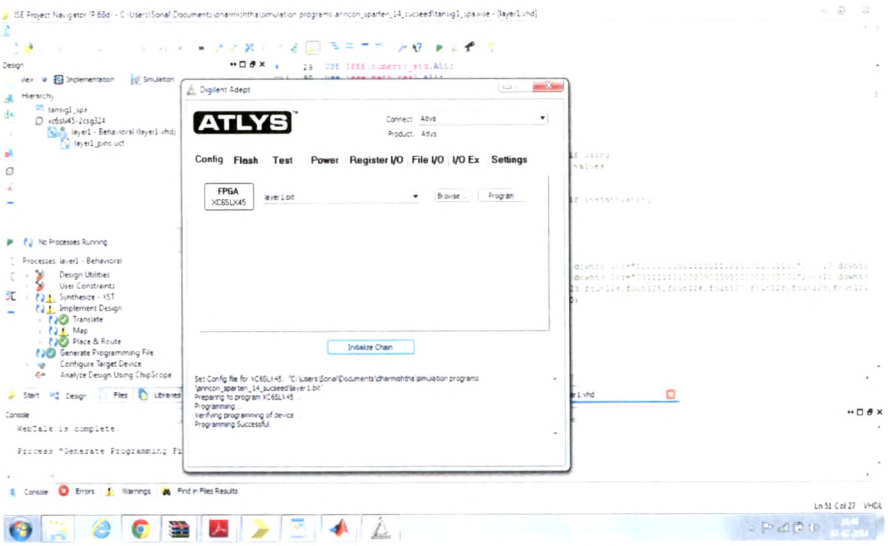

Figure 8.7: Snapshot of programming FPGA kit

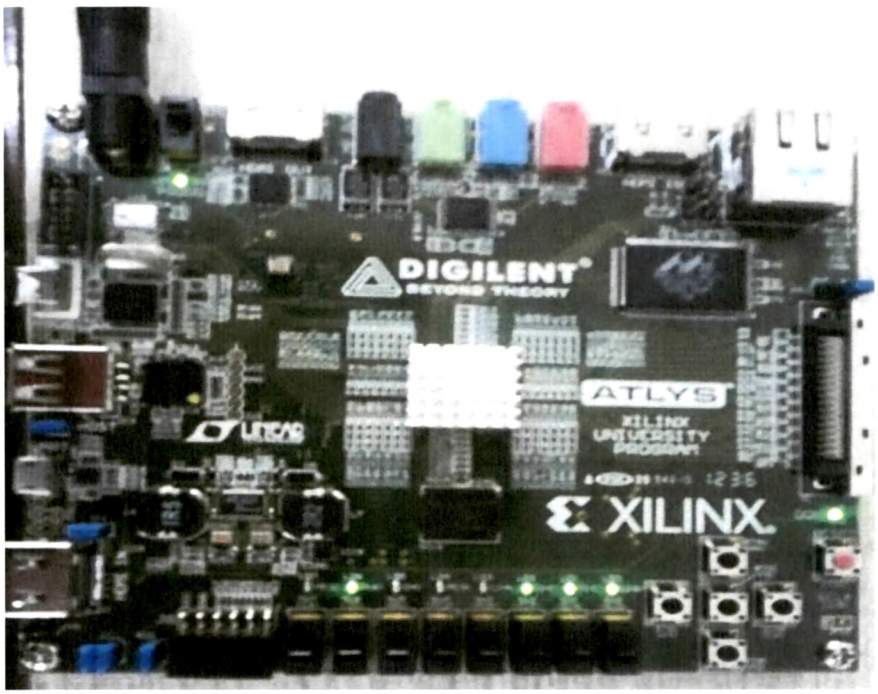

Figure 8.8: Output on LED for given combination of the inputs

SECTION B: WSN Hardware implementation

Current popular low-end wireless sensor network hardware is small sized, uses low cost Reduced Instruction Set Computer (RISC) microcontrollers and provides a small amount of program and data memory (about 100 kB). Mainly for status indication most boards integrate up to three LEDs. Many Companies such as Intel, Crossbow, Dust

Networks, Millennial Net, Arched Rock, Ember, and others manufacture Sensor network devices (motes).

The size of a wireless sensor nodes are usually varied from a shoe box size to the size of a gold coin [10]. The following Figure 8.9 shows what a typical wireless sensor node looks like. The future trend of WSN devices are going to become cheaper, smaller and longer energy lasting [11].

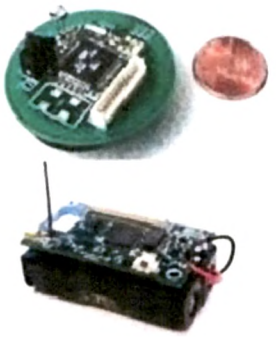

Figure 8.9: Typical wireless sensor nodes size [10]

The traditional wireless sensor (see Figure 8.10) consists of a communication device (e.g. radio transceiver/transmitter) for wireless communication, a microprocessor for processing data, sensing device (sensor board) for sensing of a physical or environment conditions, and power device (e.g. battery or solar panel) to provide the sensor nodes with the power needed [10,12, 13].

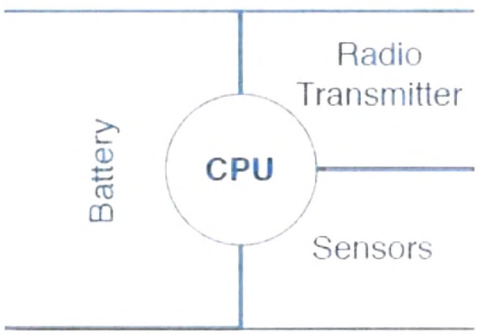

Figure 8.10: Wireless sensor node components [11]

According to the different applications of the sensor node, the function and quantity of the sensing device of the nodes are also different. It can detect temperature, humidity, acceleration, noise, light intensity, pressure, the size of moving objects, speed,

and many other physical phenomena in which the observer may be interested [10, 12, 14, 15].

The hardware features of the Mote Processor Radio (MPR) platforms and Mote Interface Boards (MIB) for network base stations and programming interfaces. It is intended for understanding and leveraging Crossbow's Smart Dust hardware design in real-world sensor network, smart RFID, and ubiquitous computing applications.

8.5 CROSSBOW MICAZ (MPR2400) MOTE Processor

MICAz [16] is the latest contribution to the Mica family evolution. Mica, released in 2001, was carefully designed to serve as a general platform for wireless sensor network research. Mica2, the successor to the Mica platform, was released one year later and corrected several of Mica's shortcomings. In 2004 MICAz was released and replaced the Chipcon CC1000 radio with the CC2420, an IEEE 802.15.4 compatible radio.

MICAz uses the Chipcon CC2420 radio in the 2.4 MHz band, a wideband radio with O-QPSK modulation with DSSS at 250kbs. The radio's higher data rates allows for shorter active periods and thereby reducing energy consumption. The CC2420 provides a number of hardware accelerators to achieve better performance. These include encryption and authentication, packet handling support, auto acknowledgments, and address decoding. The MICAz is capable of establishing and maintaining a multi-hop mesh network. The MICAz is used in this thesis for sensor-to-gateway communication.

Figure 8.11 below shows the general layout of the MICAz. The MicaZ specifications are described in [18].

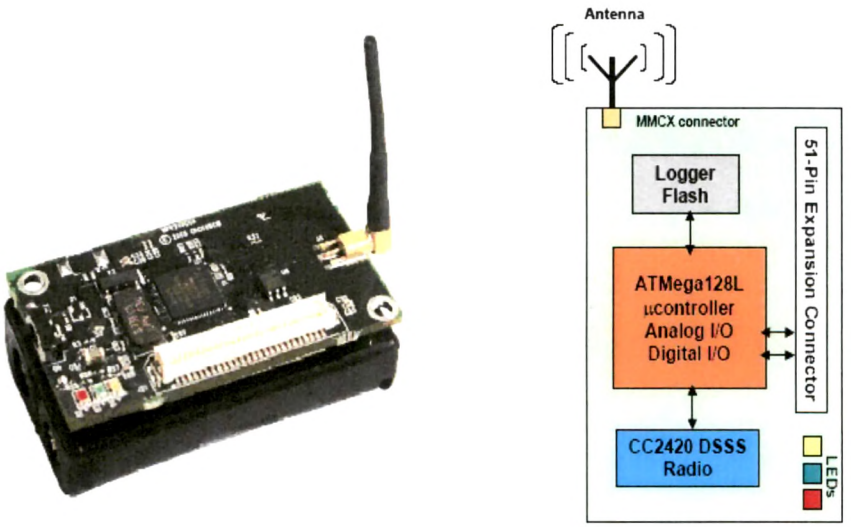

Figure 8.11: MICAz mote [courtesy Crossbow]

8.5.1 CC2420 radio transceiver

The CC2420 RF transceiver is mounted on the MPR2400 board for the purpose of wireless communication. It is a single-chip 2.4 GHz IEEE 802.15.4 compliant RF transceiver designed for low power and low voltage wireless applications [17]. CC2420 includes a digital direct sequence spread spectrum (DSSS) baseband modem providing a spreading gain of 9 dB and an effective data rate of 250 kbps. The MicaZ's CC2420 radio can be tuned from 2.048 GHz to 3.072 GHz which includes the global Industrial, Scientific and Medical (ISM) band at 2.4 GHz. IEEE 802.15.4 channels are numbered from 11 (2.405 GHz) to 26 (2.480 GHz) each separated by 5 MHz.

The CC2420 provides one very important piece of metadata about received packets. This is received signal strength indicator (RSSI), which is a measurement of the power in dBm present in a received radio signal. It is calculated over the first eight symbols after the start of a packet frame. RSSI can also be sampled at other times, to detect the ambient RF energy. RF transmission power is programmable from 0 dBm to -25 dBm. Typically, the CC2420 consumes the current of 18.8 mA in the transmit mode and that of 17.4 mA in the receive mode and have a typical sensitivity of -95 dBm.

8.5.2 MIB520 USB interface board

The MIB520, shown in Figure 8.12, provides USB connectivity to the MICA family of Motes for communication and in-system programming. It supplies power to the devices through USB bus.

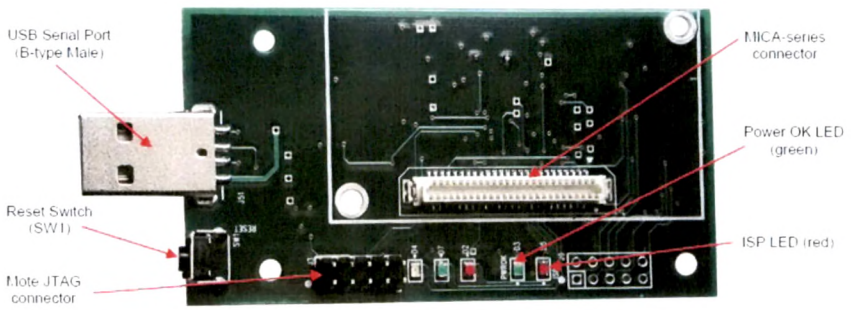

Figure 8.12: Photo of top view of an MIB520CB

The MIB 520 has an on-board in-system processor (ISP) – an ATmega16L to program the motes. Code is downloaded from a PC to the ISP through the USB port. Next the ISP programs the code into the mote. The mote which is attached to the MICA-series connector of the MIB520 is defined as the base station. It allows the aggregation of sensor network data onto a PC. Any MicaZ mote can function as a base station when it is

connected to the MIB520. Therefore, the MIB520 provides a fundamental serial/USB interface for both programming and data communications for any WSN.

8.5.3 MDA100CA/MDA100CB

MD100CA and MDA100CB (shown in Figure 8.13) have the same content except for some minor changes. The MDA100 series sensor boards have a precision thermistor, a light sensor/photocell, and general prototyping area. The prototyping area supports connection to all eight channels of the Mote's analog to digital converter (ADC0–7), both USART serial ports and the I^2C digital communications bus. The prototyping area also has 45 unconnected holes that are used for breadboard of circuitry.

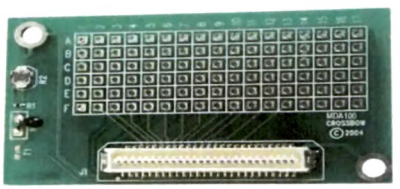

Figure 8.13: MDA 100CB

8.5.4 TinyOS

TinyOS [18] is an open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture, which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks. TinyOS's component library includes network protocols, distributed services, sensor drivers, and data acquisition tools—all of which can be used as-is or be further refined for a custom application. TinyOS's event-driven execution model enables fine-grained power management yet allows the scheduling flexibility made necessary by the unpredictable nature of wireless communication and physical world interfaces.

TinyOS has a component-based programming model (codified by the nesC language). Like other operating systems, TinyOS organizes its software components into layers. The lower the layer the closer it is to the hardware; the higher the component, the closer it is to the application. A complete TinyOS application is a graph of components, each of which is an independent computational entity.

Components have three computational concepts: 1) commands, 2) events, and 3) tasks. Commands and events are mechanisms for inter-component communication, while tasks are used to express intra-component concurrency. A command is typically a request to a component to perform a service. A typical example is starting a sensor reading. By

comparison, an event would signal the completion of that service. Events may also be signalled asynchronously, for example, due to hardware interrupts or message arrival. From a traditional OS perspective, commands are analogous to downcalls and events to call backs. Commands and events cannot block. However, a request for a service is split-phase in that the request for service (the command) and the completion signal (the corresponding event) are decoupled. The command returns immediately and the event signals completion at a later time.

Rather than performing a computation immediately, commands and event handlers may post a task, a function executed by the TinyOS scheduler at a later time. This allows commands and events to be responsive, returning immediately while deferring extensive computation to tasks. While tasks may perform significant computation, their basic execution model is run-to-completion, rather than to run indefinitely; this allows tasks to be much lighter-weight than threads. Tasks represent internal concurrency within a component and may only access state information within that component. The TinyOS scheduler uses a non-preemptive, first in, first out ("FIFO") scheduling policy. For more details on TinyOS and nesC programming concepts, refer to the "TinyOS/nesC Reference Manual" by Phil Levis included on the *MoteWorks* CD.

8.6 Software Description and Discussion

This section describes the software provided by the manufacturer for programming the motes.

8.6.1 Software Development Tools

MoteWorks™ [18] is the end-to-end enabling platform for the creation of wireless sensor networks. The optimized processor/radio hardware, industry-leading mesh networking software, gateway server middleware and client monitoring and management tools support the creation of reliable, easy-to-use wireless OEM solutions. OEMs are freed from the detailed complexities of designing wireless hardware and software enabling them to focus on adding unique differentiation to their applications while bringing innovative solutions to market quickly.

MoteWorks is provided with a set of software development tools for custom Mote applications, including custom sensor board drivers, sensor signal conditioning and processing and message handlers. MoteWorks includes an optimized cross-compiler for the target mote platform and an advanced editor for TinyOS application development. MoteWorks automatically installs and configures these development tools for quick set-up and rapid start of development.

Within the MoteWorks framework a minimum of five files will be placed in any application's directory:

1. Makefile
2. Makefile.component
3. Application's configuration written in nesC
4. Application's module written in nesC
5. README (optional)

Figure 8.14 (a) and (b) shows the method of executing .nc file and programming motes by writing the following command on Tools>shell.

Make micaz install,0 mib520,com10

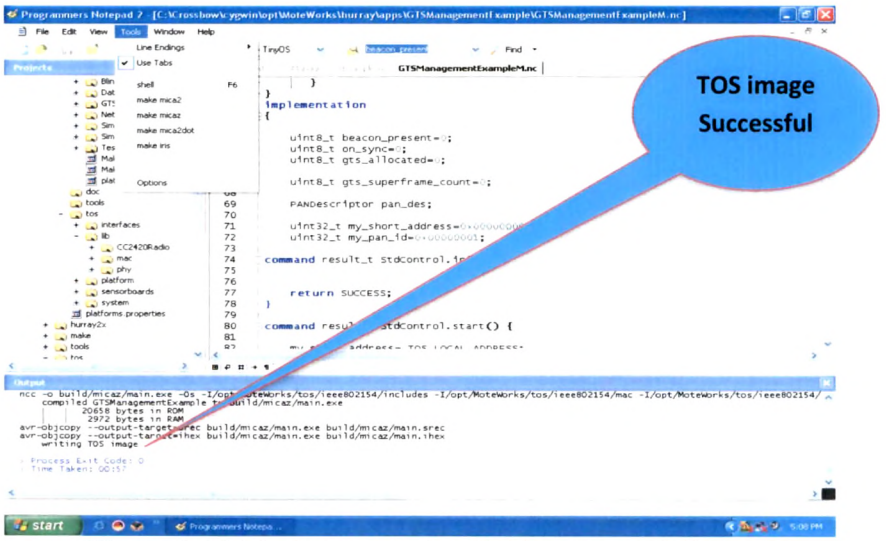

Figure 8.14 (a): Programming Environment of Motes

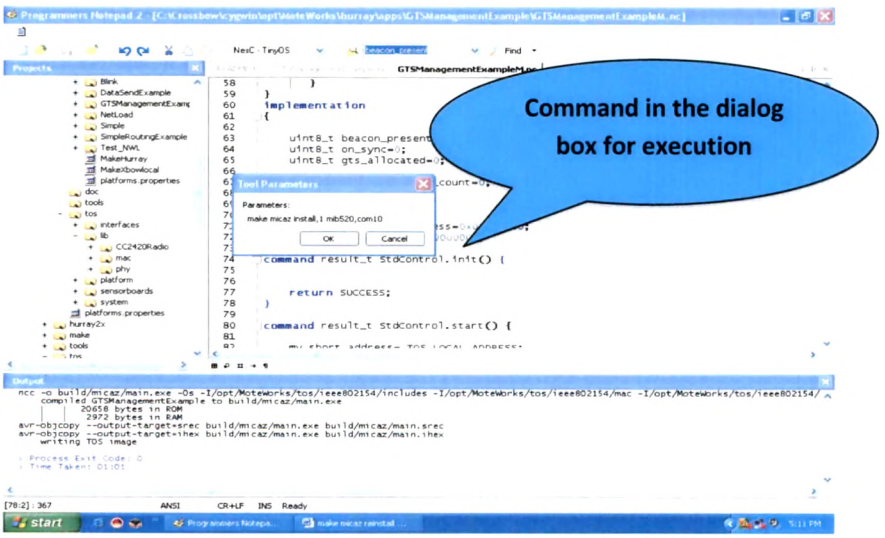

Figure 8.14 (b): Programming Environment of Motes

After the compilation has completed one should see "writing TOS image" as the last line in the Output window shown in Figure 8.14(a), otherwise it shows error in one of the files. After the successfully loading the programme in mote one can see the message "Uploading: flash" shown in Figure 8.15.

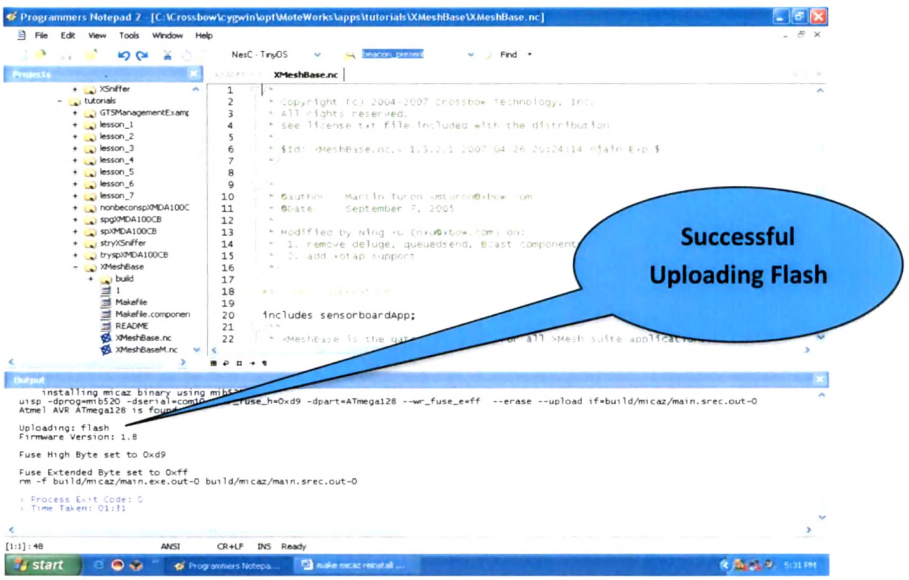

Figure 8.15: Snapshot of successful programming done in motes

8.6.2 MOTE-VIEW Functionalities

MoteView [19] is designed to be an interface between a user and a deployed network of wireless sensors. MoteView provides the tools to simplify deployment and monitoring. It also makes it easy to connect to a database, to analyze, and to graph sensor readings. The key function of the program is to monitor the communications between the gateway and the individual motes.

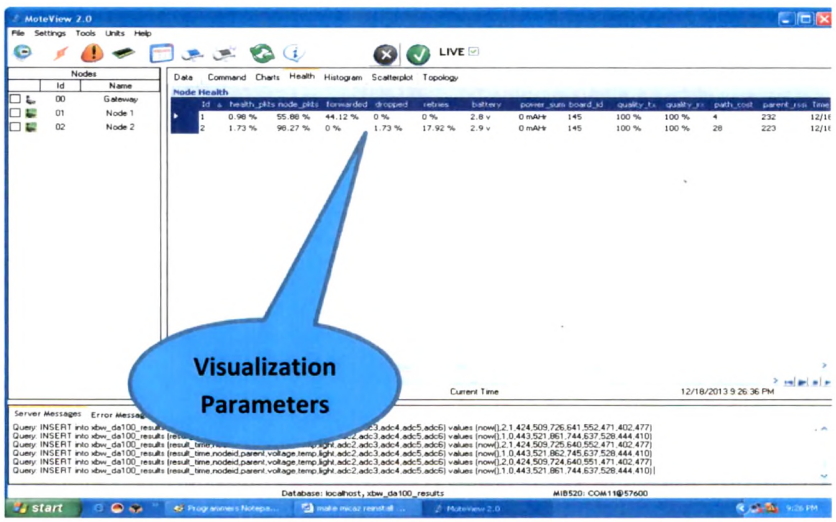

Figure 8.16: Screenshot of the database in Health view

The data can be displayed using the MoteView program. The color of the mote icons on the left hand side of the program's graphic user interface (GUI) indicates the overall health of the connection from the mote to the gateway. The green color indicates the signal is good and the latest signal received from the particular mote is current. Figure 8.16 shows the visualization of parameters. This screen can be accessed by selecting the Tools icon on the menu bar and selecting the Program Mote option. [19]

8.7 Test-bed Hardware Setup and Implementation

In this section, the entire network nodes are built on MICAz platform– MICAz XMDA100 WSN starter kit from CROSSBOW which includes three sensor nodes and one base station.

8.7.1 Program Sensor Nodes

This subsection explains test-bed on IEEE 802.15.4 WSN Star network operating in beacon enabled mode, with one PAN Coordinator and one End Device. Both real test-bed and the simulation will be set with the same initial parameters.

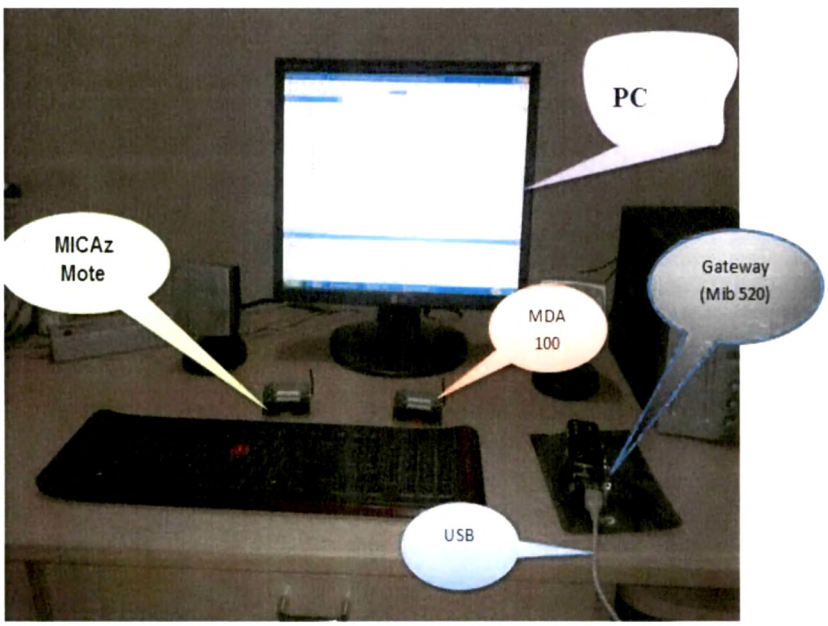

Figure 8.17: Experimental Test bed using MICAz Motes

Figure 8.17 depicts the experimental test-bed using MICAz motes. When the Coordinator node is turned on, the end node synchronizes with its beacon and starts transmitting data frames with the respective configurations.

Performance metrics were defined in order to evaluate the performance of the beacon enabled mode. These metrics are means of comparison between experimental and simulation results. The simulation and the experimental scenarios are depicted in Figure 8.18(a) and (b), respectively.

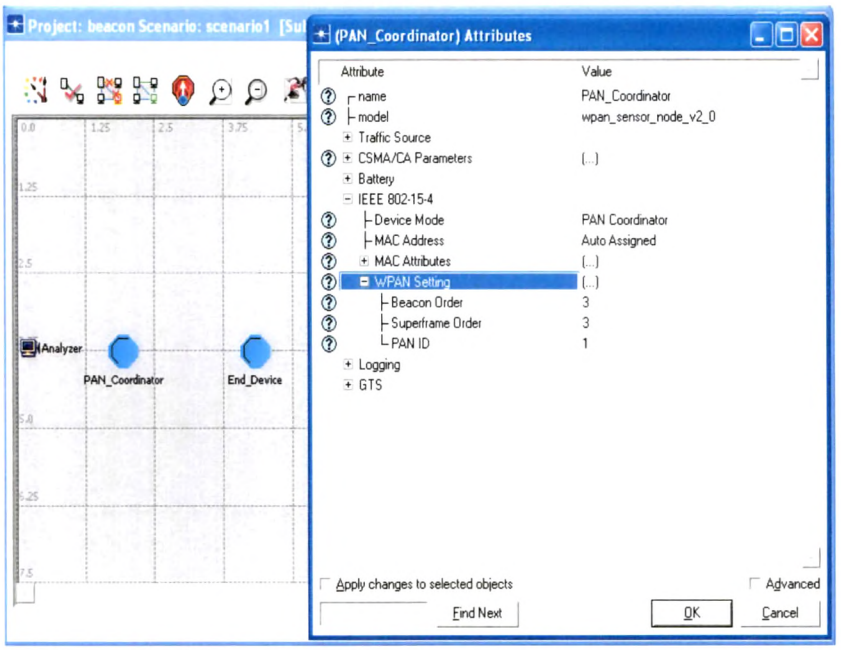

Figure 8.18(a): Simulation Test bed

Figure 8.18(b): MICAz Nodes & Gateway Setup

Figure 8.19: Snapshot of connecting mote on gateway through USB port

Figure 8.19 shows the snapshot of connecting mote on gateway and CPU USB port to programming the motes and Figure 8.20 shows the red LED on during successful uploading programme.

Figure 8.20: Snapshot of uploading programme

8.7.2 Experiment Results

Figure 8.21 shows the results of the node throughput of test-bed and simulation with same scenario. The nature of the graphs are similar for test-bed experiment and simulation i.e., for the traffic load, the node throughput decreases as the data rate increased. The results shown in test-bed experiments are considered in the four rounds experiments.

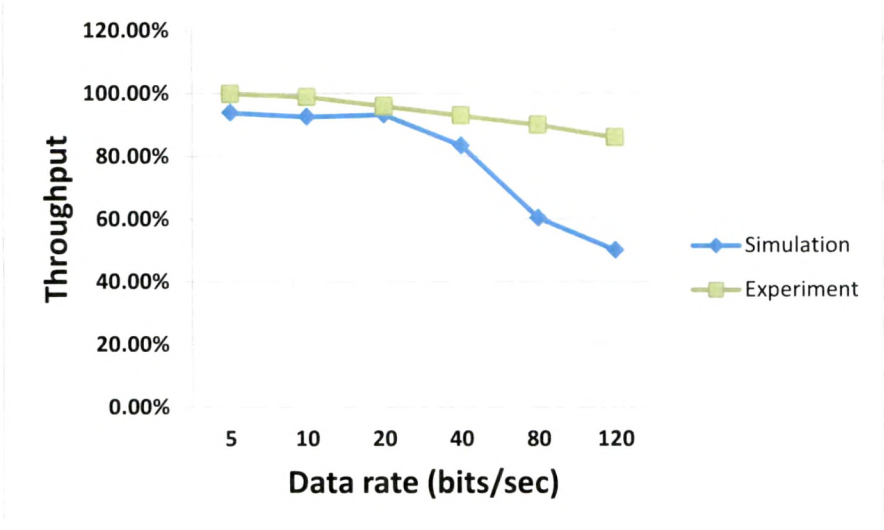

Figure 8.21: Test bed versus Simulation results

As it can be seen from the graph, nature of data rate versus throughput is same for both test-bed experiment and simulation; there is a small difference in both the results. The reasons cause the results distance between simulation and test-bed is that behaviour can be the increased number of failures due to higher medium congestion and the simulation default setting uses low RSSI and considers GTS (Guaranteed Time Slot) in the scenario.

Summary

This chapter describes FPGA hardware implementation of ANN configuration. Feed forward type Multilayer Perceptron (MLP) neural network with Tansig as an activation function is used to decide Packet size for given input parameters data rate and inter arrival time. Result comparison of the FPGA implementation of ANN with the Matlab implementation is done by calculating relative error. In the second section of this chapter implements real time WSN using MICAz motes with same scenario in simulation and comparison is done to get conclusion that simulation and experiment gives same behavior for same environment.