Firmware Over The Air Deployment System

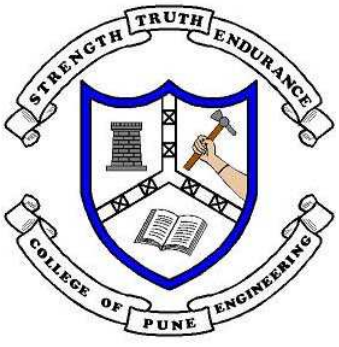
A dissertation submitted in partial fulfillment of
the requirements for the degree of

Master of Technology

by

Saurabh Kumar Panda
(MIS No. 121229015)

Under the guidance of
Mr. S. Basu
and
Mr. S. Kakade



Department of Electrical Engineering
COLLEGE OF ENGINEERING–PUNE
22 June 2014

I dedicate my work in this project to my loving family and friends.

Dissertation Approval

The dissertation entitled

Firmware Over The Air Deployment System

by

Saurabh Kumar Panda
(MIS No. 121229015)

is approved for the degree of

Master of Technology

_____	_____
Examiner	Examiner
_____	_____
Guide	Coguide

_____
Chairman

Date: _____

Place: _____

COLLEGE OF ENGINEERING PUNE, INDIA

CERTIFICATE

This is to certify that dissertation entitled **Firmware Over The Air Deployment System** submitted by **Saurabh Kumar Panda** (MIS No. 121229015) is a record of bonafide work carried out by him, under my guidance, at College of Engineering, Pune in partial fulfillment of the requirement for the award of the degree of **Master of Technology** with specialization in **Control Systems** from University of Pune.

Date:

Place:

Mr. S. Basu	Mr. S. Kakade	Dr. S. S. Dambhare
Adjunct Faculty	Asst. Professor	Head of the Department
Electrical Engg.	Electrical Engg.	Electrical Engg.
College of Engineering	College of Engineering	College of Engineering
Pune - 411005	Pune - 411005	Pune - 411005

Abstract

With advancements in science and technology, hardware and software developments help handle and manage wireless systems in more reliable and efficient ways. Huge networks involving large number of wireless sensor nodes spread over an extensive area and synchronize their operations with few centralized controlling nodes called routers. In an industry using wireless nodes, identifying and resolving defects in the firmwares of those nodes is a tedious task and may lead to heavy downtime and losses. Firmware Over the Air technology for wireless embedded systems provides a smarter and faster solution by allowing the firmwares of the nodes to be updated or replaced by a new and better firmware without having physical connections between the router node (node sending the firmware over the air) and the targeted node.

The project aims to establish a system which will enable firmware over the air (F.O.T.A.) technology to be deployable in an already implemented embedded wireless sensor network. In this project the firmwares developed have been successfully tested on real-time system involving two nodes (router-node and target-node) and a computer where the user operating the computer is provided with basic F.O.T.A. functionality options like read, write and verify flash of a physically disconnected node. The developed radio driver responsible for point-to-point communication between two nodes was tested stable and the radio was configured to default receive-mode. The firmware developed for the router established a communication link between the computer port at the user side and wireless module at the target-node. An user interface was created to control the entire over the air burning process which could repeatedly fire low-level commands successively to the target-node boot-loader. The target-node firmware upon receiving series of low-level commands performed simple tasks which added-up together to complete a high-level task as intended by the user. The work carried out in the project may be directly deployed to wireless star-networks involving platform-nodes specifically used in this project. This project-report discusses the idea, approach and the implementation details of the project while also mentioning its future scope and extensions.

Contents

List of Tables

List of Figures

Abbreviations

WSN	Wireless Sensor Network
FOTA	Firmware Over The Air
ADC	Analog to Digital Converter
HEX	Hexadecimal
PC	Personal Computer
OS	Operating System
RAM	Random Access Memory
EEPROM	Electrical Erasable Programable Memory
NRWW	No Read While Write
RF	Radio Frequency
GUI	Graphic User Interface
TI	Texas Instruments
ACK	Acknowledgment
FIFO	First In First Out
RX	Receive
TX	Transmit
MCU	Micro-Controller Unit
BL	Boot-Loader
MOSI	Master Out Slave In
MISO	Master In Slave Out
CSn	Chip Select

Chapter 1

Introduction

Advancements in communication techniques and embedded systems technologies have made researches look for new platforms of data collection and processing. There is an urge to make network operations involving a huge area of coverage and large quantities of data simpler to control and monitor in a channelized way. In huge wireless networks large number of wireless nodes spread over extensive areas and synchronize their operations of data collection and sharing along with few centralized controlling nodes. Today, in advanced wireless systems communication between the nodes is one of the many functions of a wireless node. To make the entire system adaptive to changes and keep them bug-free the nodes need to constantly update their firmwares. However, updating the node firmware over the air (through wireless communication) will make the system easy to monitor and save money, time and manpower.

Maintenance of any embedded system is as big problem as establishing it. Typically in a real life scenario if a breakdown occurs in an established wireless network system it takes days to identify and fix the problem. First day generally is the day of evaluation of the problem involving an engineer, while another day or two is needed for a specialized engineer to work on the specifics of the problem identified and the final day for setups and some testings to ensure error-free working. This breakdown not only requires time to solve but also involves a lot of manpower and monetary costs. Firmware over the air feature makes the maintenance of system simpler and faster if any software problem arises by allowing the nodes to replace or upgrade their firmwares without having to establish any physical connections with the controlling nodes. In a system with F.O.T.A. feature fault detection and the repairing could be made wireless. This project aims at developing such a system.

1.1 Wireless Sensor Networks

The building blocks of a Wireless Sensor Network (WSN) are wireless sensor nodes, which are responsible for collection of data and communicating them to other nodes. A sensor node generally constitutes of four major components:

1. a sensing unit having following sub-units:
    - sensors and
    - analog to digital converters (ADCs)
2. a processing unit
3. a transceiver unit
4. a power unit

Research in WSN field is primarily concentrated in improving the performance and efficiency of the above mentioned units. Today the wireless systems have almost completely replaced their wired counterparts. The reasons become obvious when we see the advantages a wireless system provides over a wired system.

Wired and Wireless Networks

The wired systems may require low initial cost of deployment but wireless systems are easy to maintain and may be easily centralized to gain control over a huge network from a single point of operation. If the network is very dense and spread over a large area it is not desirable to go for a wired connection as it complicates the situation even more. Today, the wireless nodes have reduced in sizes and are able to embed in them powerful processors that can carry out heavy duty tasks. The wireless nodes have shown better performance in adverse environmental conditions where manual placing of nodes with wired connections are impossible. The merits of switching to a wireless systems from wired-systems overcome the con-factor of initial cost of deployment and manufacture.

Table 1.1: Wired vs. Wireless comparison

	Wired	**Wireless**
Connection	Physical conn. reqd.	No physical conn. reqd.
Security	High security. Less Interference	Less secured. High Interference
Deployment Cost	Low	High to Very High
Bandwidth	High	Low
Maintenance	easy to maintain small networks	Larger networks easy to handle

1.2 Motivation

1.2.1 Practical Applications of WSNs

Properties like robustness, deployability, dependability, scalability have introduced and empowered use of wireless technology in almost every field of application technology. To understand the importance of this project it is necessary to know the fields which have been benefited by use of Wireless Sensor Networks.

1. Security and surveillance: Sensor nodes with motion sensing capabilities may be deployed at the borders to detect the intruder crossing the line of control.

2. Environmental monitoring and remote sensing.

3. Scientific research and investigation.

4. Threat-Identification: Sensors can be used to identify potential threats such as chemical contamination of water distribution system at various locations, pathogens in the environments, and other subtle changes in critical infrastructure.

5. Disasters response: Wireless sensor networks are also found useful for detection of various disasters such as landslide, volcanoes and forest fire.

6. Automation of industries: The cabling required to be done to connect various sensors in any industry can be redundant by using wireless sensor nodes. This simplifies placement and controlling of the sensors resulting in more accurate measurements and centralized control.

7. Smart Dust: Probably the most ambitious application in the field of research in WSN field comes as SMART DUST. Size reduction is paramount to make the nodes as inexpensive and easy-to-deploy as possible while still achieving impressive performance in terms of sensor functionality and communications capability. Advancements in hardware technology and engineering design have led to reductions in size, power consumption and cost. This has produced compact, autonomous and advanced nodes that contain multiple sensors, are capable of high computation and require minimal power to operate. These millimeter-scale nodes which currently find its application in military and security purposes are called Smart Dust. It is certainly within the realm of possibility that future prototypes of Smart Dust could be small enough to remain suspended in air, buoyed by air currents, sensing and communicating for hours or days to provide necessary information.

1.2.2 Firmware Over The Air (F.O.T.A.) in WSNs

In an already deployed wireless system there maybe a huge number of nodes operating synchronously over a large field to collect data, process it and use them to control the system or the surroundings. The nodes thus form an essential and integral part of the entire WSN system and it is necessary that they function well irrespective of external factors. Many a times the entire system fails to achieve its desired goal because of malfunctioning of some nodes. The nodes may suffer from hardware or software limitations listed below.

- **Hardware Defects and Limitations:**
    1. components get worn out
    2. batteries die out
    3. hardware is obsolete and out-dated
- **Software Problems:** An important part of the sensor nodes is the embedded micro-controller which operates according to a predefined program burnt onto it. Problems may arise in the program that was burnt onto the micro-controller. Some common problems are:
    1. malfunctioning of program or bugs found in the code.
    2. data corruption in the program.

3. program not efficiently utilizing the resources available.
4. program is incompatible and non-adaptive to newer versions of hardware and software.

It is difficult to rectify hardware problems without physically accessing the nodes but the software problems can be tackled by remotely accessing the program part of the nodes i.e. using the **Firmware Over The Air** (F.O.T.A.) feature to change the application code of the micro-controller. To access the program memory of any micro-controller and change or re-burn its application code generally a physical connection is needed between the micro-controller and an **External Programmer**. However, for wireless nodes a smarter way to do the burning process is by attaching a transceiver to the micro-controller and burning its flash memory over the air. This idea is referred to as firmware over the air (F.O.T.A.) and using this feature in WSN adds a new dimension to wireless technology and eases the control and maintenance of the entire system. Some note-worthy advantages are:

- The nodes are sometimes placed in physically inaccessible areas.
- The nodes in some application projects are very small, huge in number and get easily mixed-up with the surroundings. So, it is difficult to individually wire them up to have a manual setup to re-burn their micro-controllers.
- A single operator can complete the entire job of changing the codes of many nodes, sitting at a single place saving manpower, time and money.
- For maintenance in some plants, accessing the nodes manually would require shutting down of certain sections of the plant which may cause heavy monetary losses. This can be easily avoided if the nodes are up-graded over the air which would ensure minimal downtime and faster, easier recovery.

Hence, F.O.T.A. in a WSN is not an absolute necessity but the idea of making a system wireless is totally futile if for every small software changes physical access to the nodes is required.

1.2.3 Scope of Research

While research and developments in hardware areas of WSN provide more flexibility and ease of use, the software and protocol researches primarily aim to increase efficiency of the overall system. The key areas of research in WSN field are:

1. identifying and developing alternate power sources.
2. distributed detection techniques.
3. multi-hop protocols, scheduling, time synchronization and coverage.
4. data mining and data handling.
5. very low cost and very low power mixed signal design of the WSN radio chip.
6. reducing size and cost of sensors for different applications.

It is important to note that F.O.T.A. has been a common feature for smart phones' operating system up-gradations but very little work has been reported for embedded systems. Building a reliable application oriented system to implement this feature for micro-controllers may be taken up as project and research in the field of WSNs.

1.3 Project Statement & Description

The previous section of the chapter suggests the motivation and the importance of having F.O.T.A. system deployed in a WSN. This project is application-oriented and aims at providing an user interface which will provide options to the user to read and write the flash memory of the target node's micro-controller and control the entire procedure.

Project Statement: To develop "Firmware Over The Air" deployment system for a Wireless Sensor Network.

It may be important to note that the project primarily aims at implementing and establishing the idea of F.O.T.A. to embedded systems (taking specific hardware components), while refining the codes and making them platform independent may be taken up as subsequent projects.

1.4 Overview of Chapters

The **first chapter** of this report provides an introduction to the the topic of Wireless Sensor Networks, its application and the research works currently being carried out in this field. It also mentions the importance of the Firmware Over The Air feature in a WSN, emphasizing the motivation behind taking up this application project.

The **second chapter** deals with background information related to this project and better introduces the problem statement of the project. There is also a section that introduces the components of this project.

The **third chapter** describes in detail the design and the approach behind the project. It also provides the details of the firmwares driving the hardware components of the project. This is the most elaborated chapter of the report that tries to explain in detail the idea and its implementation.

The **fourth chapter** named "Tests and Results" briefly mentions the tests that were needed to be done to check for bugs and code-flow of the firmwares developed. The results are provided through various snapshots. A section lists out various statistics related to the project obtained though this tests.

The **fifth chapter** provides future scope of the project. This is the final chapter of the report and thus concludes the report on "Firmware Over The Air Deployment System".

Chapter 2

Background

2.1 Defining the Approach to the Project

To update the contents of a micro-controller, a trivial method is to connect the MCU with a computer (generates HEX file and controls the process flow) through an external programmer. The extra component **external programmer** is not needed in case of wireless burning of the micro-controller in F.O.T.A. process. The concept of F.O.T.A. is tried, tested and implemented for smart-phone applications and hence the concept is not new to the world. Here, we try to extend this idea to micro-controllers in a wireless sensor node. In a general wireless system huge number of nodes are deployed. Although all the nodes are responsible for communication there are also few special nodes dispersed through-out the network called **router**s. In a general network, a router is a centralized controlling node communicating data with many wireless terminal or intermediate nodes. The routers in turn may be connected to some powerful processing device like a **computer** which analyzes the received data and decides the data to be communicated. In a WSN with F.O.T.A. feature, a router is able to change the firmware of any sensor node and the working of the router is in turn controlled by the processing unit (computer) connected to it. It is to be noted that for a WSN to develop this feature the sensor nodes or the target nodes whose firmwares need to be changed must be pre-loaded with another special firmware, the **boot-loader** which will carry out two major tasks:

1. establish wireless communication with the router
2. provide functions to read and write the flash memory

2.2 Components of the Project

Being an application project it is expected to involve many hardware and software components for its implementation. In this section, a brief introduction to some of these components is provided so that the purpose, idea and the implementation of the project is very well understood.

2.2.1 Hardware Components

Motes

Motes are wireless nodes responsible for communication. They form the building blocks of any wireless network. The micro-controller and the radio (transceiver) are two important sub-units of a mote. In this project we need a router (Mote-1) and a target (Mote-2), which are structurally identical to each other but carry different pre-loaded program burnt in their micro-controllers. Essentially, the **router** sends command and data signals to change the firmware inside the **target mote**. Practically, the following mote platforms are generally used to design a WSN. The size of these motes varies from roughly the size of a matchbox to the size of a pen tip. These motes have been nicknamed Smart Dust. Example Motes: Rene, MicaZ, IRIS, SHIMMER, TelosB, Sun SPOT. The Table 2.1 compares these platforms while Figures 2.1 and 2.2 are the pictures of the **motes** used in this project.

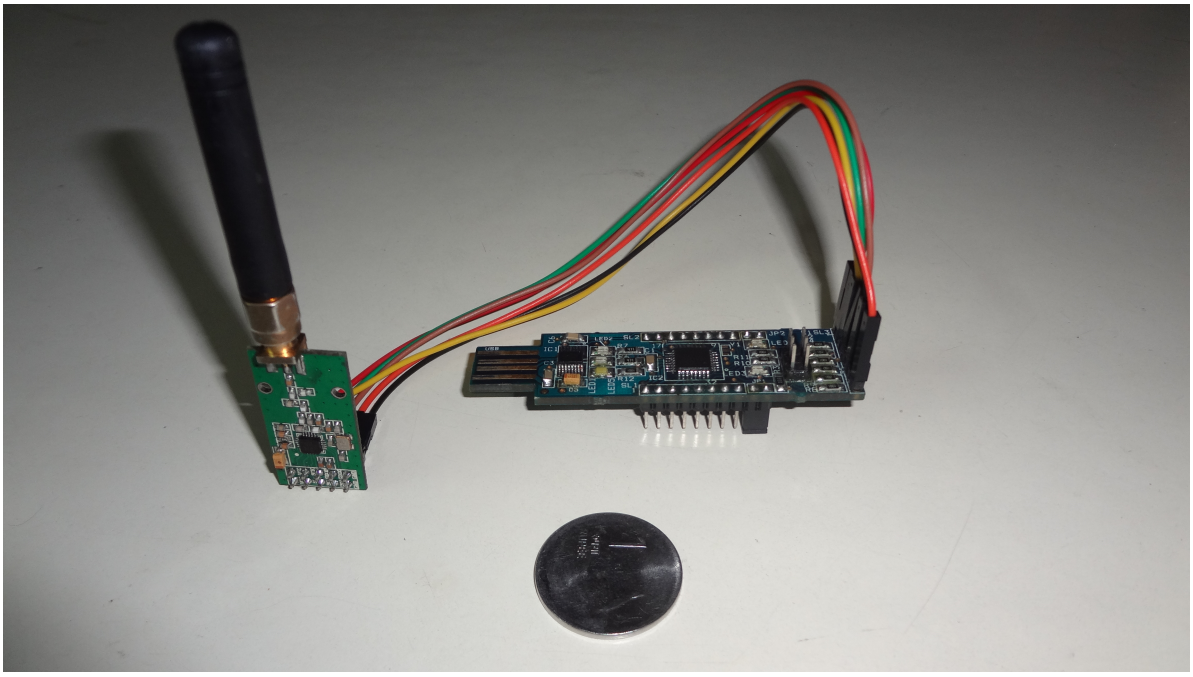


Figure 2.1: Hardware Mote used for the project showing the radio and the micro-controller board

Table 2.1: Comparison of different Mote Platforms

Mote Name	Microcontroller	Tranceiver	Program+Data Memory	External Memory	Programming	Cost (US $)	Remarks
Rene	ATMEL8535	916 MHz radio with bandwidth of 10 kbit/s	512 bytes RAM	8 KB flash		100	TinyOS Support
MicaZ	ATMEGA 128	TI CC2420 802.15.4/ZigBee compliant radio	4 KB RAM	128 KB flash	nesC	99	TinyOS, SOS, MantisOS and Nano-RK Support
Iris Mote	ATmega 1281	Atmel AT86RF230 802.15.4/ZigBee compliant radio	8 KB RAM	128 KB flash	nesC	115	Mote Runner, TinyOS, MoteWorks Support
SHIMMER	MSP430F1611	802.15.4 Shimmer SR7 (TI CC2420)	48 KB flash 10 KB RAM	2 GB microSD Card	nes C and C Programming	269	TinyOS Support.
TelosB	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10 KB RAM	48 KB flash		99/139	Contiki, TinyOS, SOS and MantisOS Support
SunSPOT	ARM 920T	802.15.4	512 KB RAM	4 MB flash	Java	750	-

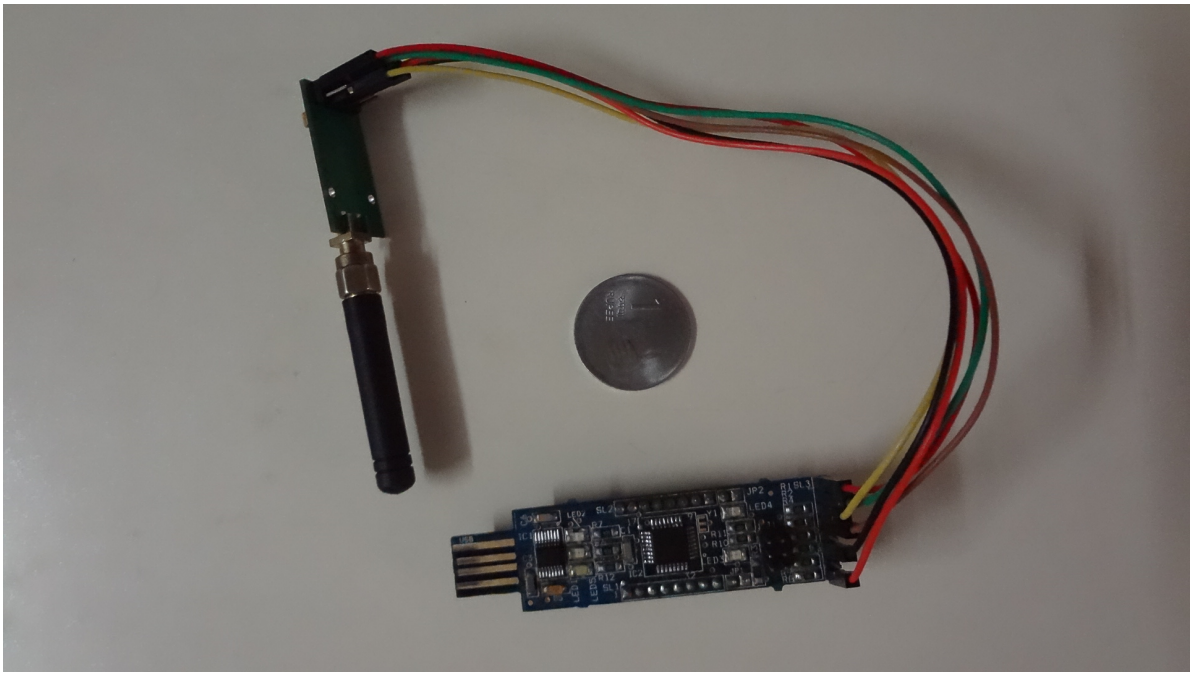


Figure 2.2: A picture of Hardware Mote (radio + micro-controller board) with ₹ 1 coin as reference

P.C.

An **user interface** resides in a computer which serves as a connecting link between the external world (the user) and the router. The router is connected to the computer via serial port and the user interface in the computer controls the entire process by sending bytes of data to the router. The P.C. also serves as an output device that displays the acknowledgement and errors that indicate the flow of the entire process. In this project to utilize the user interface firmware available the P.C. should have a LINUX O.S. (Operating System) and LUA programming language installed in it.

Micro-controller

The micro-controller is the brain to the motes in WSNs. It not only controls the functioning of the radio attached to it but also performs other tasks like to collect data from sensors, operate ADCs and if needed do processing of the received data. Some important hardware specifications considered while choosing ATmega328P micro-controller for this project are:

- Self-programming: This feature enables a micro-controller to burn data in its RAM to its own flash memory.
- Flash memory size = 32KBytes: The Flash Memory of target mote should be large enough to support the firmwares of core boot-loader, radio stack and the application program combined.
- RAM size = 2KBytes: High RAM gives extra liberty while developing firmware in creating and maintaining data variables.
- EEPROM size = 1KBytes: EEPROM stores data values even after the micro-controller RESETs.
- N.R.W.W. section in flash = 2KBytes: It is recommended that the boot-loader read and write flash functions should lie in the special N.R.W.W. (No read while write) section of the flash memory. This is the only section where read flash or write flash calls may done.

Figure 2.3 shows the ATmega328P micro-controller board used in the project (with FTDI chip for USB-serial communication and other accessories) while Figure 2.4 provides the details of the board.

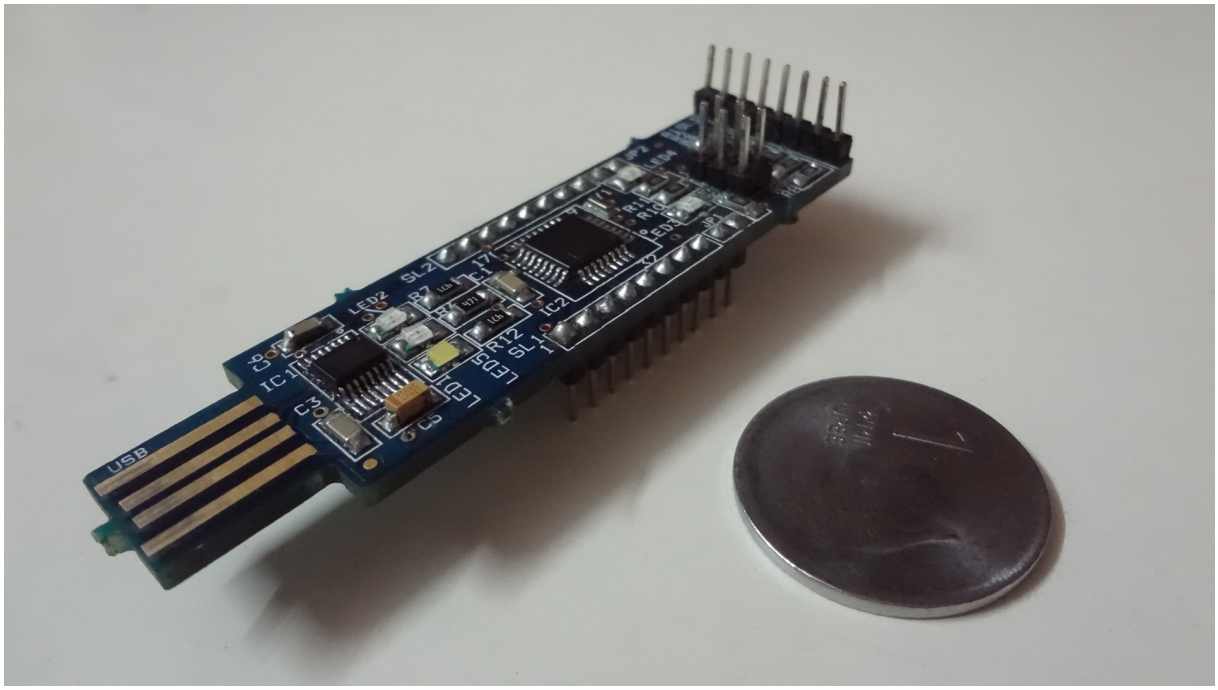


Figure 2.3: Picture of the micro-controller board used in the project with reference to ₹ 1 coin

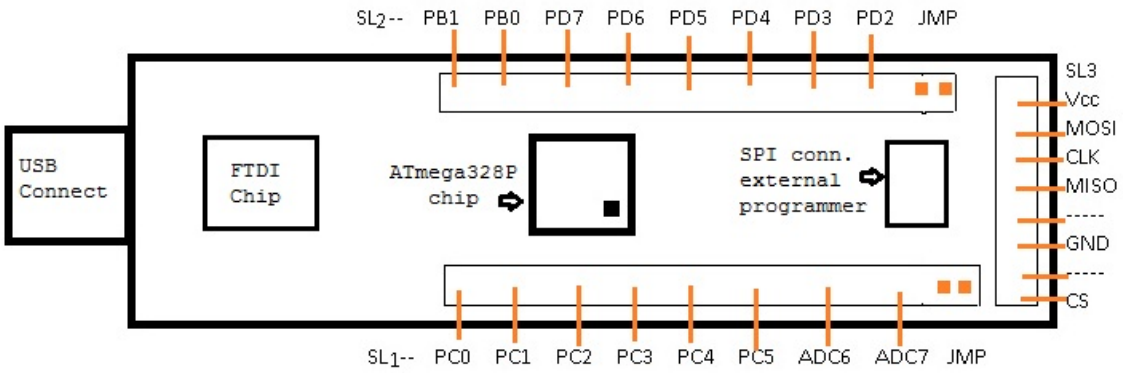


Figure 2.4: The pin-configuration and board details of the MCU board

Radio

This is the transceiver connected to a processing and a power unit which is responsible for wireless communication. The state-flow and operation of the radio is controlled by the processing unit attached to it. In this project sub-GHz transceiver TI-CC1101 radio is used which has an RF frequency range upto 1GHz. Features of CC1101:

- Low Power Consuming

- Frequency Bands: 310MHz to 348MHz, 420MHz to 450MHz, and 779MHz to 928MHz
- Power Output: 10dBm to 12dBm
- Data Rate Maximum: 250kbps to 600kbps
- Range: 700m nearly

2.2.2 Software Components

AVR Accessories

Some open-source AVR software used in the project include

1. AVRstudio 4 (windows OS): used for compiling and debugging embedded C code for ATMEL micro-controllers.
2. avrdude: needed to use external programmer and burn or read flash memory of micro-controller.
3. gcc-avr: for compiling the code and generating HEX file.
4. simavr: for debugging code.

NOTE: The firmwares of the entire project were developed using LINUX O.S. and open-source software.

Lua Programming Language

LUA is a scripting language based on **C** language. It is simple but very powerful language to use for writing programs that can be developed on **C** as their programming platform. Here, in this project an user interface had to be created that can handle string operations (extract data from file and manipulate them) and also control the serial port of the computer. This interface was written in **LUA**.

TI RFstudio

RF-Studio is a software by Texas Instruments that helps in deciding the radio register settings once the configuration and its state machine have been decided. This software provides a G.U.I. to export register settings for Texas Instrument Radios.

Documenting and Code-Handling Software

- Doxygen: It is a documentation software that eases the code documentation if the code is written in **C**, HTML or other languages.
- Latex and Miktex: A documenting language used for report publishing, paper work and documentation.
- GIT: This software creates local and web repositories for the firmwares developed.

Chapter 3

Methodology

3.1 Introduction

From the earlier discussion having seen the advantages of having Firmware Over The Air (F.O.T.A.) feature in a wireless system it may be inferred that the feature adds an extra dimension to wireless communication by enhancing efficiency in terms of time, money and manpower. This project primarily aims at establishing the idea, design and the implementation of F.O.T.A. to a sensor network. Hence, for the accomplishment of the project's aim we require:

- the two motes:

  Router Mote (Mote-1)

  Target Mote (Mote-2)

- a computer with a user

In this project, point-to-point communication between the two motes was developed which was necessary to establish two-way communication between two wireless nodes. Of-course the very same idea and approach can be easily extended and applied to a large **star-connected** network, while more research on communication routing and other techniques may enable this idea to spread to any generalized wireless network. In general, a WSN with F.O.T.A. feature will have small number of routers and a huge number of the sensor nodes distributed throughout the network. The **router** generally is responsible for sending the command and data packets while the target sensor nodes loaded with the special firmware, the **boot-loader** get updated by responding to those packets. The entire process may be controlled by checking the functioning

of the router, hence it becomes highly centralized as the router in turn is connected to a P.C. (personal computer) operated by a user.

This chapter provides an insight to the idea of the project, the approach taken up and its implementation. It is important to note that if the idea of implementing the project in a small network is a success it can be extended to larger networks.The final section of the chapter deals with coding details and state machines of the firmwares developed.

3.2 The Design

A typical wireless sensor network may involve huge number of nodes with random network configurations but to establish the idea of F.O.T.A. in WSN would require two identical nodes (a router and a target node) and a computer. The physical connections of the three components and the design of the project is shown in the Figure 3.1. The user interacts with the P.C. and inputs the choices offered by the user interface like reading flash, writing flash while also viewing the corresponding acknowledgment during the process being carried out. The router communicates with the P.C. through the serial port of the computer and conveys the command and data packets to the target-mote over the air. The target mote receives the required packets and takes up necessary actions after which again sends wireless acknowledgement to the router and hence the computer, which is analyzed by the user operating the computer.

In the design, it is taken care that the computer is responsible for maximum processing task (extraction of data from HEX file, analyzing acknowledgement, maintaining process flow) while the router and the target mote are entitled with simpler tasks because of the hardware limitations they suffer. The router here acts as a simple connecting link between the wireless medium and the serial port of the computer. The target mote has been provided with predefined small tasks which are called upon request by the P.C. and are executed after which the target sends appropriate responses to the computer.

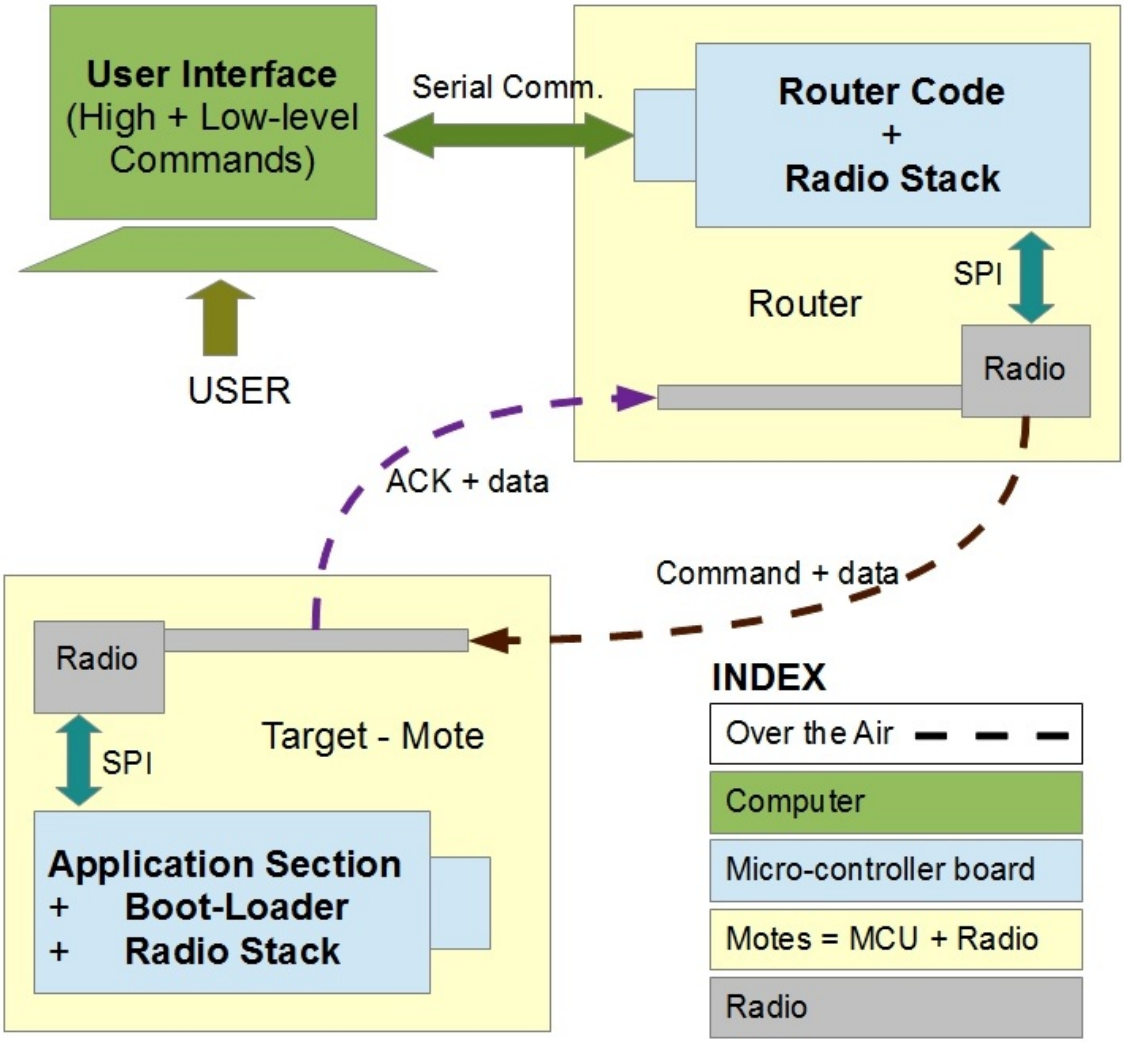


Figure 3.1: Block Diagram of the Project

3.3 The Implementation

As briefly mentioned in the earlier section, in the project structurally there are three hardware components required to achieve the goal of the project. However, the implementation of the project if made modular would rest upon four important steps that were to be cleared one after the other. The earliest milestone of the project was to build the **radio stack** or the radio driver that will enable the motes to communicate with each other. Here, point to point communication between two motes with radios of different addresses was achieved considering minimum packet loss and minimum data corruption as priority. The job of the driver was to provide user controllable functions (like initialize radio, send packet, receive packet) to control the function-

ing of the radio by the micro-controller while avoiding any deadlock.

The next step involved developing the special firmware, the **boot-loader** that was to be installed in the target mote to enable it to use the radio stack and be able to communicate with the router and according to commands and data received re-program or read the flash memory. The technique of re-positioning of the code in the flash memory of the micro-controller by modifying the **linker-script** file was implemented which enabled sectioning of the entire flash memory (see Figure 3.2) of the target mote micro-controller.

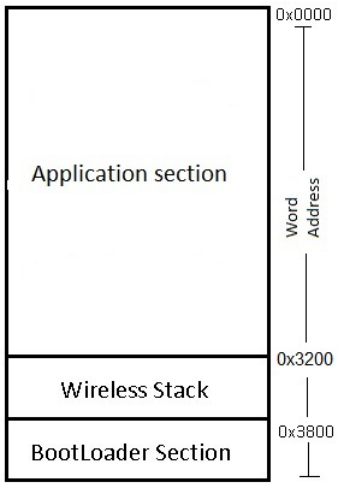


Figure 3.2: Segmentation Flash Memory of Target Mote

Following are the reasons behind relocating the codes:

- The boot-loader code was to be sectioned in specific flash memory region of micro-controller (origin address = 0x7000 in bytes) for it to start running upon a RESET to the micro-controller unit.

- The boot-loader firmware should be placed inside limited space (2KBytes for ATmega328P) of **N.R.W.W.** (No Read While Write) section of flash memory which might not be enough to include the radio stack (used by the boot-loader code to send and receive packets). Thus, the radio stack was moved and placed at a separate newly created segment "stack" (address of origin on flash = 0x6400 in bytes) to allow maximum space for the core boot-loader and avoid conflict of flash memory space.

- The third and the most important reason was to create a separate section for the application program to be stored into and keep the boot-loader and radio stack firmware safe

in the flash memory of the micro-controller. The user may burn the data from the generated HEX-file to application segment (address of origin of application segment on flash = 0x0000 in bytes) of flash and prevent corruption or overwriting in the reserved segments containing codes for boot-loader and radio stack.

The third major challenge was to develop the **router code** and the **user interface** in the computer. It is necessary to note that the router in this project is a simple transparent-mode device that builds up the connecting link between the user operating the P.C. and the target mote. The router was configured to receive only low-level commands and data from the P.C. via the serial port and re-convey them to the target mote by forming necessary radio packets, while also receiving corresponding acknowledgement from the target and reporting them back to the computer. The user interface in the computer accepts input from the user through keyboard of the computer and accordingly sends commands and data values to the target through the router. To simplify the debugging process of the entire project, the commands delivered to the router and hence to the target were **low-level commands**, which commanded the boot-loader to perform simpler tasks like filling 16-bytes of data in the temporary buffer, reading 16-bytes from the flash memory and copying the temporary page buffer to the flash memory. However, the set of **high-level commands** in the user interface was to make entire system more user-friendly and understandable. A single high-level command may trigger multiple low-level commands one after the other automatically until desired the high-level task gets completed each time receiving desired acknowledgement from the target as response.

The final step was to critically **test** the entire firmware for bugs, loop-holes, data-corruption and deadlock conditions . The entire system entered a series of automated low-level command triggering after receiving any valid input from the user. Thus, each component of the system had to **synchronize** with the activities of other components. The following section provides details about the firmware, their code-structure and state-machines of the major components of the project.

3.4 Firmware Details

3.4.1 Radio Stack

The radio communicates with the micro-controller through its S.P.I. (Serial Peripheral Interface) module. Thus, the radio driver has a basic SPI layer to facilitate data exchange between the micro-controller and the radio. To use the radio for transmitting and receiving R.F. packets (Radio Frequency packets) the highest layer of the radio provides three major functions:

1. **radio_init():** Initializes the radio to a configuration (defined by its register settings) and sets the radio to default receive mode. It also allows the user to set the radio's current address and its power setting.

2. **RFsend_packet():** This function sends the data provided as an argument by forming proper packets. The radio undergoes many state changes starting form initial RX state then to TX state, transmits the packet and then returns to default RX state.
   NOTE: Calling this function with FORCE_BIT enabled causes transmission of packet irrespective of the present state of the radio, i.e. the controller loops around in this function until the packet is sent to the RF channel.

3. **RFreceive_packet():** The purpose of this function is to check for data in the RX-FIFO (a queue) of the radio. The radio stays in default RX (receive) mode and maintains a queue for storing received packets. Hence, if the queue contains atleast one packet calling this function ensures a single packet data is copied to RAM of the micro-controller, else the function returns a rx-fifo empty error.
   NOTE: Calling this function doesn't ensure reception of a packet. It is only a check for any data in RX-FIFO.

Table 3.1 indicates the packet format that is maintained while transmitting and receiving packets:

Table 3.1: Packet format in radio communication (radio stack)

Index No.	0	1	2 + ...
Data	*length*	*destination address*	*data* + ...
size	1B	1B	1B + ...

It is important to note that the radio can be configured to operate under following modes (here we implement option-1):

1. Stay in receive mode forever and the micro-controller: can read the received packets in the queue and also sends packet when required by calling functions.

2. Stay in default IDLE mode and thus be commanded to listen to or send the radio packets whenever required.

3. Work on interrupt basis and inform micro-controller when a packet is received or transmitted.

4. Implement the wake-on-radio feature in which the radio wakes periodically from sleep mode to listen to packets, saving power dramatically.

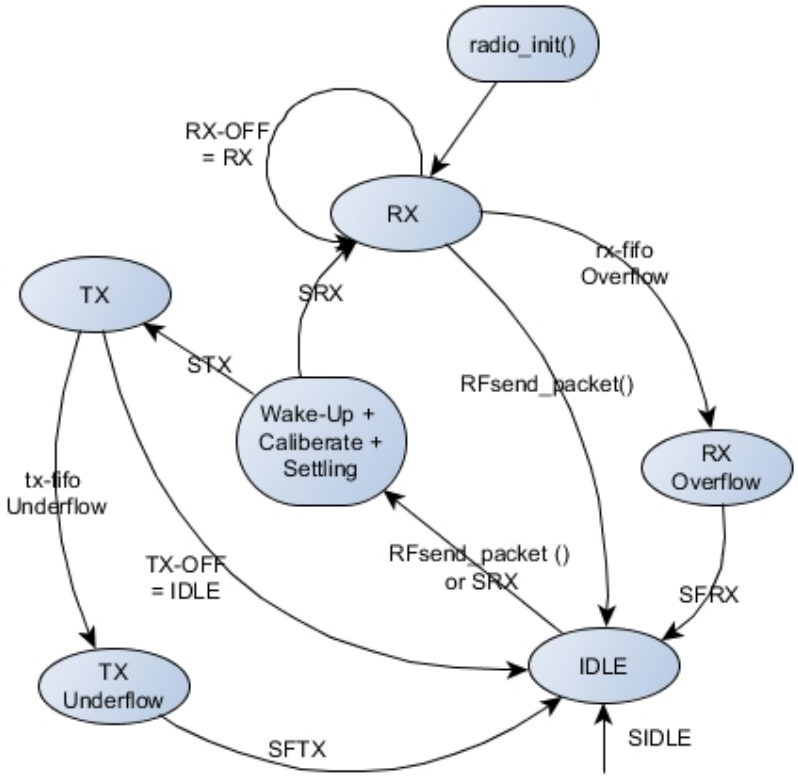


Figure 3.3: State-Machine of Radio

The Figure 3.3 shows the internal state-machine of the radio. As previously mentioned the radio constantly listens to the channel by staying in receive mode. After completing a transmit

packet task it goes to IDLE state. For overflow and underflow conditions a flush of the queue is needed which forces the radio to enter IDLE state. Whenever in IDLE state, a strobe SRX is pushed onto the radio to get the radio back to default RX state. Only after returning back to RX state can the micro-controller use send or receive functions available in radio stack.

3.4.2 The User Interface

The P.C. firmware is developed in LUA language and consists of two broadly categorized sets of commands. The user interface in the P.C. provides the user with option of inputting any high-level command and each high-level command may trigger one or multiple low-level commands until the accomplishment of the task. Each low-level command is associated with a corresponding command-byte that has to be sent through the serial-port to the router. Upon receiving back the same reflected command-byte, it is ensured to the user that the router is working properly and now is ready to receive data-bytes corresponding that command-byte. The required chunks of data are sent to the port of the computer and the P.C. awaits for acknowledgment from the target via the router. The Tables 3.2 and 3.3 provide a list of low-level and high-level commands and their functions.

NOTE: For the entire project only positive acknowledgement is implemented. Failure of reception of acknowledgement packet may indicate either of the two cases:

- incomplete task at the target mote
- packet loss or corruption of data in packet

Table 3.2: List of low-level commands used in the project

Command	**Arguments needed**	**Purpose**
fill Pg Buff 't'	Line No. of HEX-file	extracts 16-bytes of HEX-data from given line and fills temporary page buffer of target mote
flash copy 'f'	Flash Address (2-bytes)	copies entire Pg Buffer to the flash memory at flash address
read flash 'r'	Flash Address (2-bytes)	reads 16 consecutive bytes from starting Flash Address

Table 3.3: List of high-level commands in the user interface

Command	low-level comm. fired	Purpose
write ‘w’	‘t’ & ‘f’	burns entire HEX-file to target
read flash ‘r’	‘r’	reads contents of flash of target
verify ‘v’	‘r’	verifies flash data of target with HEX-file data
reset ‘o’	‘f’	resets entire process & B.L. internal variables
jump app ‘j’	*Nil*	jumps from boot-section to application section
status ‘s’	*Nil*	checks internal variables of boot-loader of target

3.4.3 Router Code

The router works as a transparent communicator which takes command (low-level commands) and data bytes from the computer, forms packet and uses the radio stack to deliver packets to the target mote and does the opposite in the reverse direction for the acknowledgment packet. The internal state-machine (see Figure 3.4) indicates the code-flow of the router firmware.

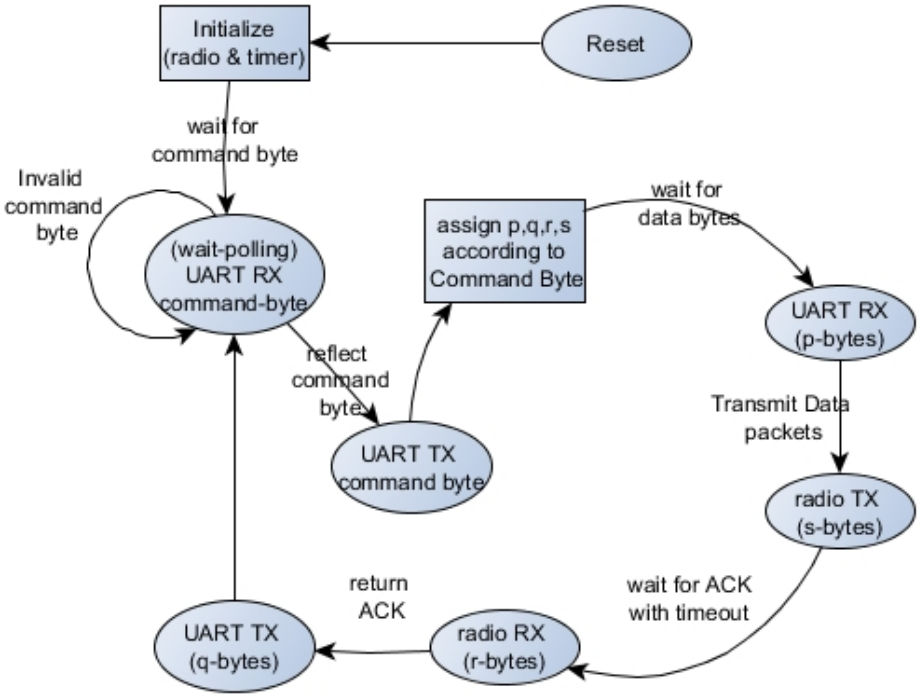


Figure 3.4: State Diagram of Router Mote (see Table 3.6)

For the router however it is important to know the number of bytes to receive or send using the radio or the serial-port interface. This is achieved by first reading the command-byte from

the serial-port and accordingly switching to form packets and expecting corresponding size of acknowledgement. The Tables 3.4 and 3.5 indicate general packet formats while sending a packet and receiving acknowledgement over the air. However, it is important to note the number of bytes in both types of packet change depending on the command-byte. The Table 3.6 indicates number of bytes of data expected to send or receive for corresponding low-level command byte.

Table 3.4: Packet format of transmitted packet by router to target

Index No.	0	1	2	3 +...
Data	*length*	*target address*	*command byte*	*argument bytes + data bytes...*
size	*1byte*	*1byte*	*1byte*	*0 or more bytes +...*

Table 3.5: Packet format of received packet by router from target

Index No.	0	1	2	3 +...
Data	*length*	*router address*	*ACK byte*	*data bytes + status bytes...*
size	*1byte*	*1byte*	*1byte*	*0 or more bytes +...*

Table 3.6: Number of bytes sent or received by router over the air for various commands

Command	**Serial TX(p)**	**Serial RX(q)**	**Radio TX(s)**	**Radio RX(r)**
fill Pg Buff ‘t’	4	16	16	4
flash copy ‘f’	4	2	2	4
read flash ‘r’	4	2	2	4
jump app ‘j’	4	0	0	4
status ‘s’	4	0	0	4

3.4.4 Target Mote Firmware

The Target-Mote firmware consists of three sections: the re-located radio stack, the boot-loader and the application section. The radio stack is used for the send packet and receive packet functions. The following are the features of the boot-loader firmware:

- The boot-loader segment begins at address 0x7000 in bytes.
- The boot-loader starts up after a reset to the micro-controller.

- As long as the boot-loader is running in target mote PB0 pin toggles almost every second and so does the LED connected to it indicating the running of the boot-loader.
- Upon receiving a special command-byte the micro-controller jumps from boot-loader section to the application section and starts executing the code at address 0x0000.
- In boot-loading mode the target mote always stays in receive mode and waits for a packet. Upon receiving a packet it extracts command-byte and executes chunk of code corresponding to the command after which it sends back acknowledgement over the air.

Figure 3.5 gives an idea about the state-machine of the target firmware while the Table 3.2 shows the tasks performed upon receiving different command bytes from user.

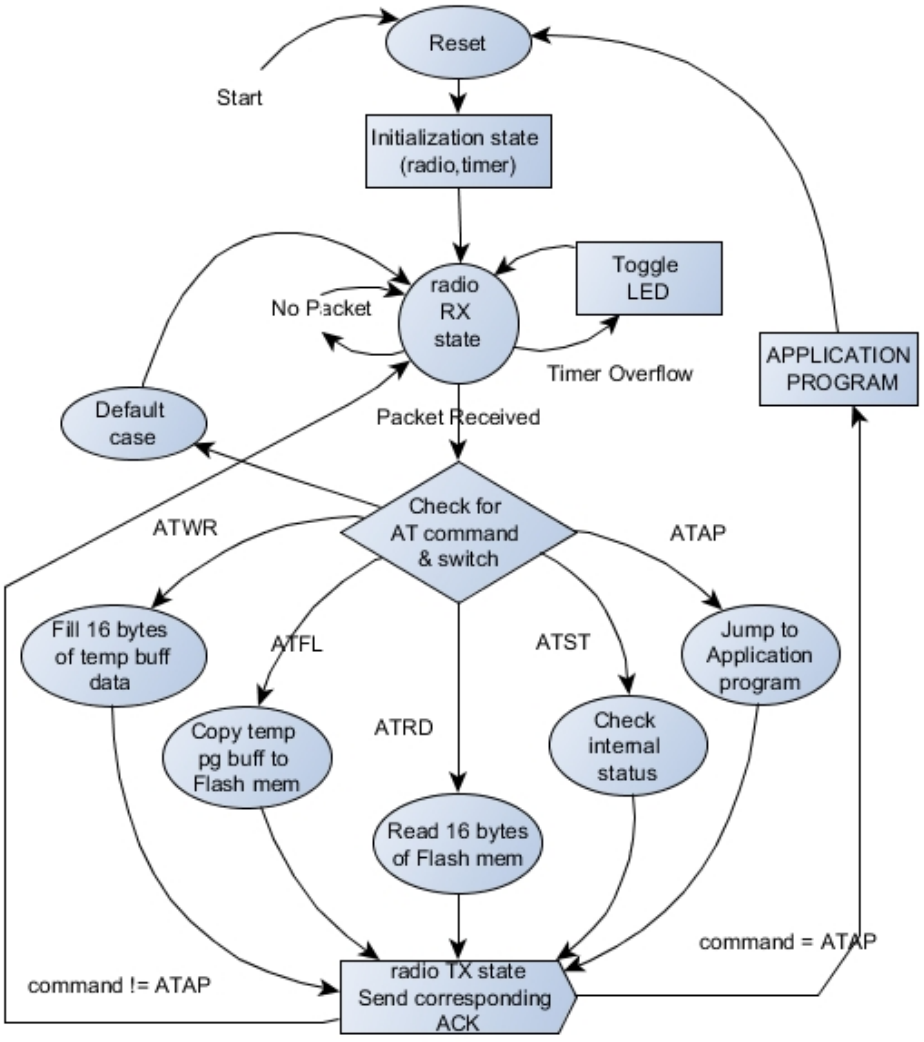


Figure 3.5: State Machine of the Bootloader

Chapter 4

Tests and Results

After the development of the firmwares, some tests were conducted to check for their appropriate working. For basic functionality at early stages small tests were done to check the code flow of the codes. However, the more these codes get used, the better are the chances of finding bugs in them and finding a solution to the problems.

4.1 The Radio Stack

The radio stack as mentioned earlier had to be re-positioned to a particular location in flash memory using the linker-script file. To check if it was actually successful we produce the "map" file while compiling the code and look for the origin of newly created "stack" segment. The following snapshot (Figure 4.1) of map file verifies that origin of the segment should be at the address of 0x6400 in bytes and the boot-loader section loaded at address 0x7000 in bytes.

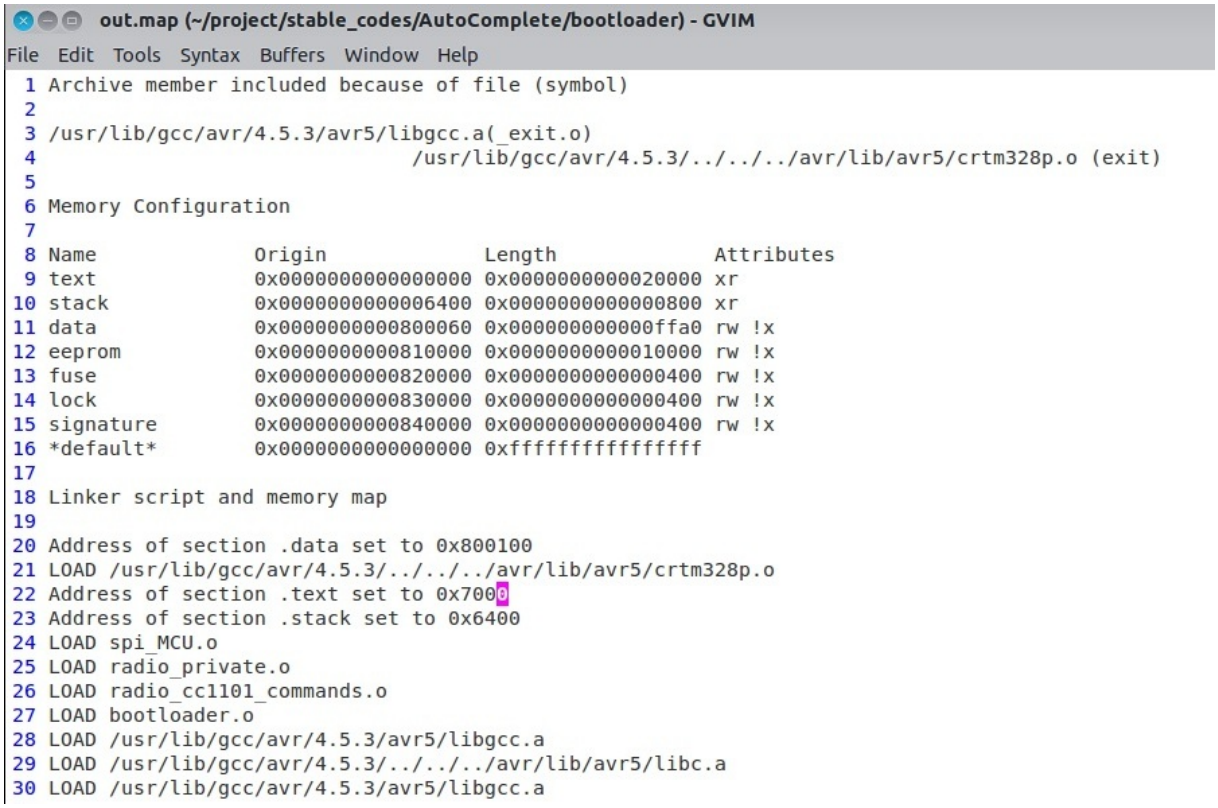


Figure 4.1: Snapshot of map file showing relocation of code

The radio stack is the basis of the project. Hence, it is absolutely imperative that the radio driver developed must avoid deadlock situations and also ensure non-corruption of packets or packet-loss. Some test-examples were taken up where the radio driver transmit and receive functions were repeatedly called in different combinations so that the radio firmware gets tested against the various real life scenarios which may arise. The two motes are programmed in such a way that:

1. Mote-1 transmits a packet and Mote-2 calls the receive function continuously.
2. Both the motes transmit a packet and enter a period of RX state alternately at different timings.
3. Mote-1 transmits packets at a much faster rate than Mote-2 calling the receive function at the other end.

The first two tests look for deadlocks and checks if packets get corrupted while the last test was to deliberately cause **RX-FIFO** overflow condition and check if the driver recovers from it. The following figures (see Figures 4.2, 4.3 and 4.4) are the snapshots from the logic-analyzer giving an idea about the transmission and reception of packets.

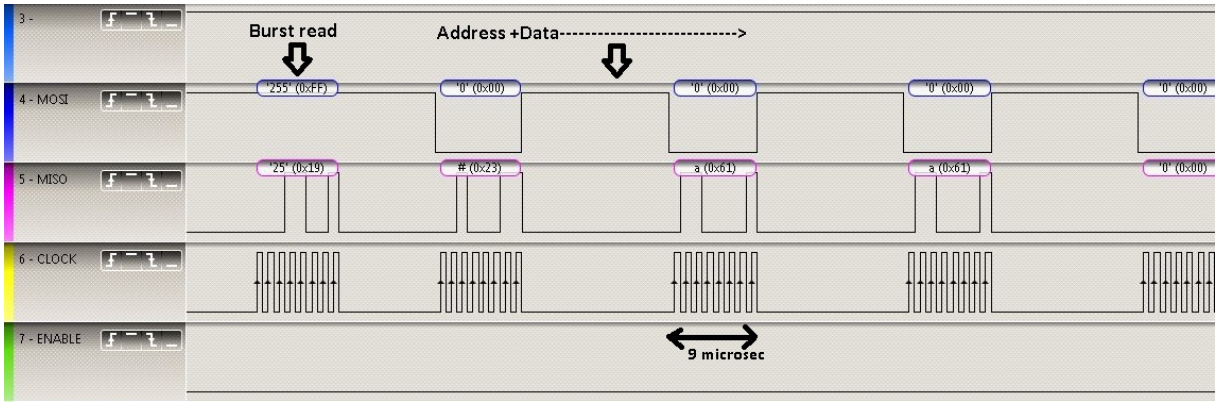


Figure 4.2: Logic-Analyzer output showing a received packet

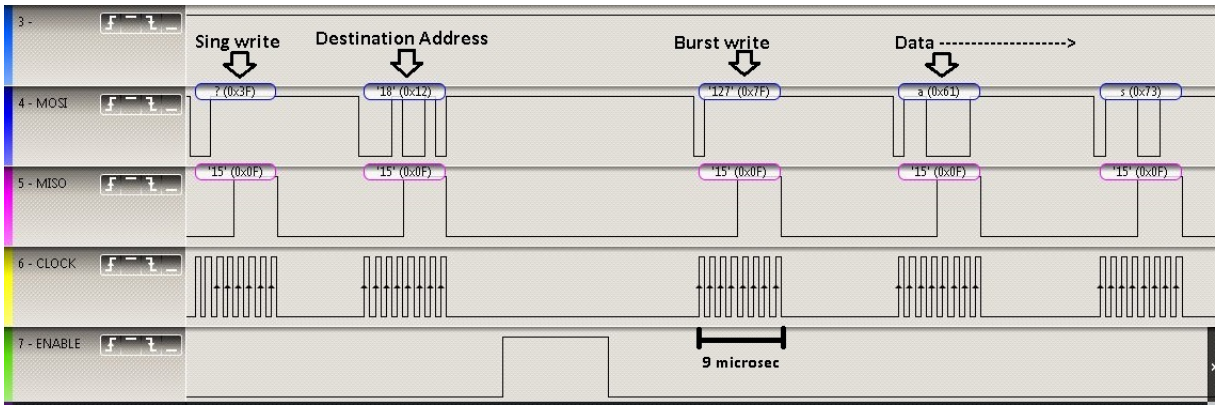


Figure 4.3: Logic-Analyzer output showing a transmitted packet

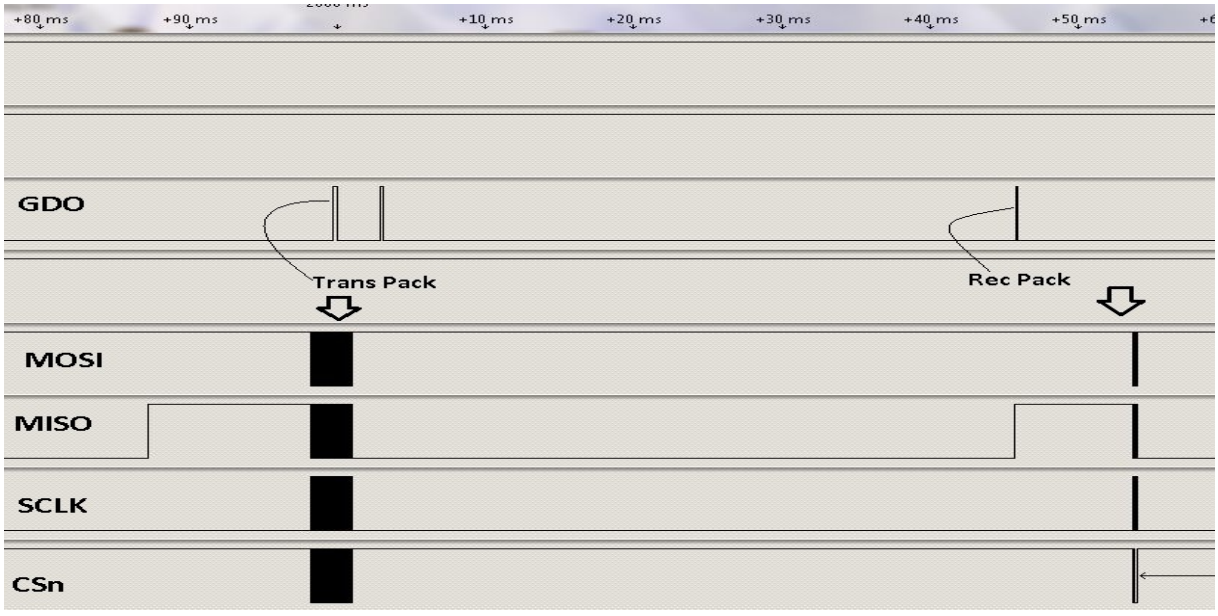


Figure 4.4: Logic-Analyzer output showing an a transmitted and a received packet

4.2 The User Interface in the P.C.

After developing a reliable radio stack it was necessary to establish two way communication between the P.C. and the target mote. A command byte was sent and it was checked if the target mote responded with an acknowledgement, the target being in reflecting mode (reflects back the received data) which indicates the user can access the target mote operation over the air and is receiving proper acknowledgement. Then each of the **low-level commands** were tested turn by turn. Finally, the high-level commands were put to test where each single command triggered a series of low-level command until completion of task. The following snapshots (see Figures 4.5, 4.6, 4.7 and 4.8) of the **terminal** shows low-level and high-level commands in the user interface, the user providing inputs and the target mote responding to them. The high-level commands may fire a successive low-level command immediately after receiving an acknowledgement from the target mote or with a small delay. The Table 4.1 gives an idea about the approximate times taken for completing various tasks with different sizes of application HEX-file generated.

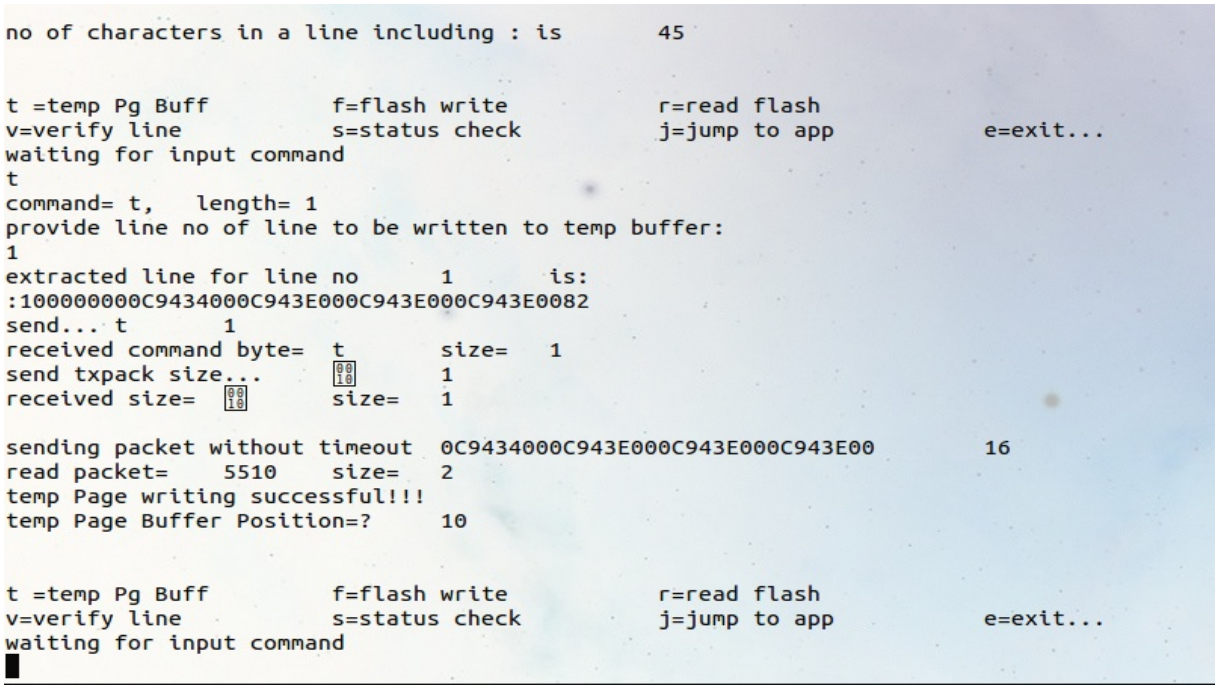


Figure 4.5: Figure showing low-level commands and **t** being the input

```
saurabh@saurabh-Amd-a10:~/project/test/PC_port_handling$
saurabh@saurabh-Amd-a10:~/project/test/PC_port_handling$ lua autoPC.lua
OK, port open with values 'device: /dev/ttyUSB0, baud: 9600, data bits: 8, parity: none, stop bits: 1, flow control: off'

input hex-file filename

file details:
       :100000000C9434000C943E000C943E000C943E0082
:100010000C943E000C943E000C943E000C943E0068
:100020000C943E000C943E000C943E000C943E0058
:100030000C943E000C943E000C943E000C943E0048
:100040000C943E000C943E000C943E000C943E0038
:100050000C943E000C943E000C943E000C943E0028
:100060000C943E000C943E0011241FBECFEFD8E04C
:10007000DEBFCDBF0E9440000C9454000C940000E1
:1000800083E084B921E092E085B1822785B985B10A
:10009000892785B94FEF50E762E0415050406040FA
:0C00A000E1F700C00000F0CFF894FFCFA3
:00000001FF



w=flash write          r=read flash           v=verify flash
o=reset from beginning         s=status              j=jump to app         e=exit...
waiting for input command
```

Figure 4.6: Figure showing high-level commands provided in user interface

```
file details:
       :100000000C9434000C943E000C943E000C943E0082
:100010000C943E000C943E000C943E000C943E0068
:100020000C943E000C943E000C943E000C943E0058
:100030000C943E000C943E000C943E000C943E0048
:100040000C943E000C943E000C943E000C943E0038
:100050000C943E000C943E000C943E000C943E0028
:100060000C943E000C943E0011241FBECFEFD8E04C
:10007000DEBFCDBF0E9440000C9454000C940000E1
:1000800083E084B921E092E085B1822785B985B10A
:10009000892785B94FEF50E762E0415050406040FA
:0C00A000E1F700C00000F0CFF894FFCFA3
:00000001FF



w=flash write           r=read flash            v=verify flash
o=reset from beginning          s=status                j=jump to app          e=exit...
waiting for input command
r

reading adress  0000
sending packet without timeout  0000    length=2
read packet=    550C9434000C943E000C943E000C943E000010  size=   19

reading adress  0010
sending packet without timeout  0010    length=2
read packet=    550C943E000C943E000C943E000C943E000020  size=   19

reading adress  0020
sending packet without timeout  0020    length=2
```

Figure 4.7: An example of high-level command **r** being input

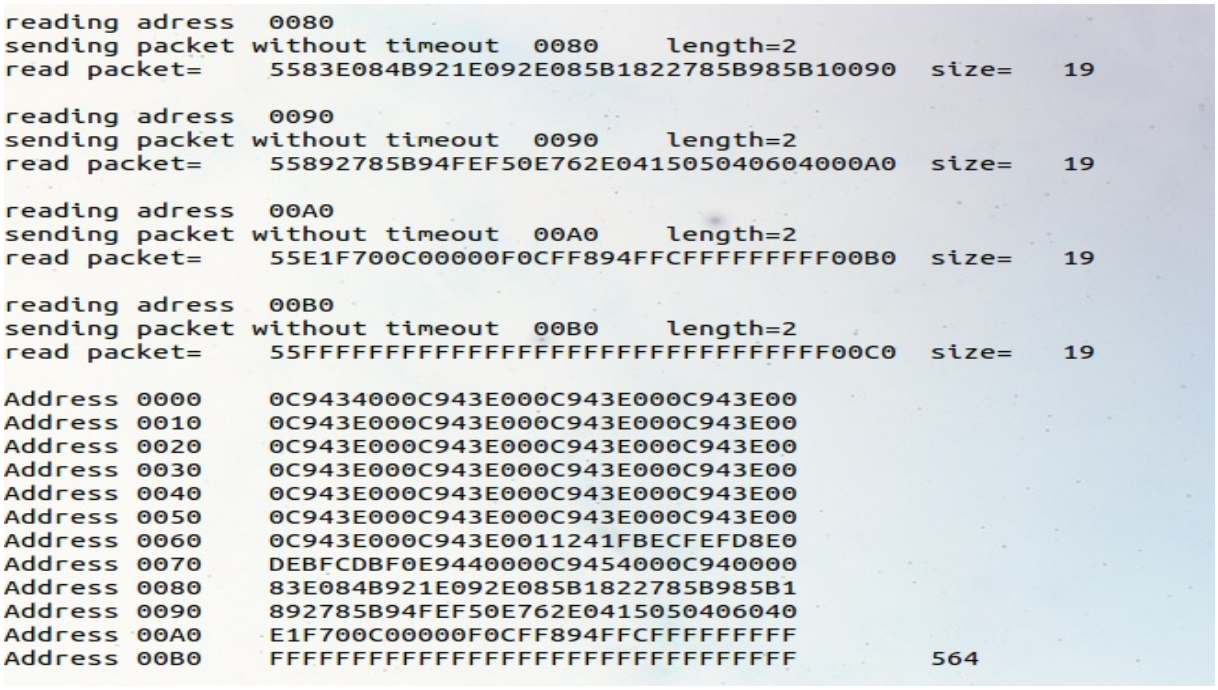


Figure 4.8: Result of inputting high-level command **r**

Table 4.1: Time taken to complete high-level tasks if low-level commands are fired without delay

Size (in bytes)	**No. of lines in HEX-file**	**Time (in msec)**				
		check status	write	read	verify	reset
172	12	3.4	456.2	428.6	398.5	12.1
1824	120		5067.5	4584.6	4528.4	
3152	197		8432.8	7513.7	7442.5	
successive firing interval		31 to 39				

Chapter 5

Conclusion and Future Scope of Work

5.1 Conclusion

Every embedded system project needs rigorous testing and extensive use of the firmwares to obtain better results. The results to the tests mentioned in the previous section suggest:

- The radio stack is stable and the mote is able to communicate without ending up in deadlock but transmitting a packet takes nearly 3ms of time which may not be optimum for many hard-time applications.
- The basic form of the Firmware Over the Air is successfully implemented on specific mote platform, where the user is provided with most necessary options of read, write and verify flash.

5.2 Future Scope of Work

The future scope of the work carried out in this project are immense, some of the extensions to this project are:

1. The problem of utilizing more time than expected to transmit a packet can be solved by analyzing the state-machine of the radio implemented now and slightly changing the flow.
2. The application HEX files having HEX-data at discontinuous addresses in consecutive lines may face corruption when using the existing process. This is a problem of extracting the data from HEX-file which can be handled by managing the LUA code well.

3. The application section has been reserved for writing the application program through FOTA process but in real life situation the process may get interrupted before reaching its completion. The mote then becomes completely useless as the the previous application firmware is overwritten while the new application firmware is incomplete. This problem needs to be taken care off. A straightforward solution is preparing two application sections and they be updated alternately where one becomes section for new program to be burnt into while the other be saved as backup in-case of incomplete FOTA process. However, the interrupt vectors in AVR micro-controllers are at beginning of flash memory, which can't be used by the second application section. Thus, using interrupts in application program with FOTA process will not be possible in this case. The solution to this problem comes as redirecting the interrupt vector tables and managing them efficiently.

4. The project at this stage is very platform specific and some effort will be needed upon changing the micro-controller or the radio. A future project can be carried upon to make the methodology and implementation of FOTA totally platform independent.

5. The user now is provided with very basic options of accessing flash memory and modifying it. There may be additional possible options that may be provided in the user interface, a note-worthy option being to be able to connect to any desirable radio address option. Various neighbor discovery protocols can be applied and then connection between nodes may be established. Also by using good routing protocols wireless burning of non-immediate neighbors can also be achieved. However, this would result in bloating-up of the existing boot-loader code which again would require optimizing code for size and managing flash segments.

6. Currently the established method is efficient for star-networks. Developing the same for mesh networks and networks with random configuration will be a major challenge.

Appendix A

Logic Analyzer Diagrams

The device Logic Analyzer is used to record pin level-changes with respect to time. In this project the device is connected to the SPI port of the micro-controller, an interface through which the micro-controller (the Master) and the radio (the Slave) communicate by exchanging single byte-value data at a time. The Logic Diagrams in the report show *five* channels: *four* SPI channels and *one* channel for GDO. The following lists the channels and the data shown in them.

1. **MOSI:** The data or strobes sent from the micro-controller (Master) to the radio (Slave).
2. **MISO:** Slave out pin indicates the data or responses from the slave to the master.
3. **CSn or ENABLE:** Chip Select with negative logic. To enable the radio chip select line has to be low.
4. **Clock:** The Clock triggering the SPI communication starts upon getting the ENABLE pin low.
5. **GDO:** A special pin from the Radio configured to indicate successful transmission or reception of a packet and hence may be used as an interrupt trigger source to the micro-controller.

Appendix B

Glossary

1. **Wireless Sensor Network:** A wireless network involving multiple sensor nodes spread over specific area of coverage.
2. **Firmware Over The Air:** An advanced feature in intelligent wireless systems where the processing unit of the system up-grades its own flash memory by receiving data through a wireless device over the air.
3. **Sensor Nodes:** Small electronic devices with a sensing and a controlling unit.
4. **Motes:** Sensor nodes available in market or generally used.
5. **Router:** A communicating-node with advanced hardware specifications which is connected to multiple other nodes. Here, the Router is the one sending the firmware (data from HEX-file) over the air.
6. **Target-Node:** The mote whose firmware is to be updated over the air by receiving data sent by the router.
7. **Boot-Loader:** A special short firmware in processing units which execute upon start-up or reset and is responsible for creating a platform for main application start-up and initialization of basic hardware peripherals. Here, the boot-loader is inbuilt for the target mote and enables it to receive wireless data and burn them onto the micro-controller while also read flash memory data and send it over the air.
8. **Radio Stack:** It is the radio driver for a micro-controller that enables it to control the functioning and state flow of the radio attached to it.

9. **External Programmer:** A physical device serving as a link between the computer and the micro-controller and is responsible for receiving serial binary data from computer and burn them onto flash of micro-controller.

10. **Application Section:** A separate section of flash reserved for the main application program of the micro-controller to be run by the user. Over the air program is written onto this section keeping other sections safe.

11. **Self Programming:** A feature available in some micro-controllers that enables them to self-modify their flash with values from their RAM while running.

12. **N.R.W.W. section:** The No Read While Write special section of flash only where it is safe to call self programming functions of the micro-controller.

13. **HEX-file:** A binary file (low level language) generated from high level language by a compiler whose data is to be written onto the flash of a micro-controller.

14. **S.P.I. protocol:** A serial communication protocol involving serial data exchange between single master and one or more slaves.

15. **Low-level Commands:** Here, these commands command the boot-loader to do very specific tasks like fill temporary buffer with 16-bytes of data, read 16-bytes of data from flash memory, copy entire buffer to flash memory etc. The entire system of F.O.T.A. in this project is based on low-level commands.

16. **High-level Commands:** A set of commands available to the user each of which fires multiple low level commands to complete a high-level task like write HEX-data from a HEX-file onto flash memory, read entire flash memory, verify the data in flash memory data with data of HEX-file.

17. **Logic Analyzer:** A device that can detect changes in levels of hardware pins, analyze and display the data value sent or received during communication.

References

[1] Vijay Rao and Subrat Kar "Design of a Novel Light-weight Hardware Module for Wireless Programming of Resource-constrained Embedded Systems", *ICCCE 2012, 3-5 July 2012, Kuala Lumpur, Malaysia*

[2] Gang Yao and Wu Zhang, Jinlin Wang "The Design and Implementation of Online-Update on Embedded Devices", *2008 International Conference on Computer Science and Software Engineering*

[3] Marko Malajner, Karl Benkic, Zarko Cucej "Online programmable wireless sensor node for testing purpose", *50 th International Symposium ELMAR-2008, 10-12 September 2008, Zadar, Croatia*

[4] Rashmi Parthasarathy, Nina Peterson, WenZhan Song, Ali Hurson and Behrooz A. Shirazi "Over the Air Programming on Imote2-based Sensor Networks", *43rd Hawaii International Conference on System Sciences - 2010*

[5] Data sheet "8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash ATmega48PA, ATmega88PA, ATmega168PA, ATmega328P"

[6] Data Sheet of cc1101, "Low-Power Sub-1 GHz RF Transceiver"

[7] Brad Schick and Cliff Lawson "Bootloader FAQ", *article Last updated: May 19, 2009*

[8] Intel Hexadecimal Object File Format Specification, *Revision A, 1/6/88*

[9] Self Programming,*article AVR109*

[10] C functions for reading and writing to Flash memory *article AVR106*

[11] Otilia Anton, Brice Gelineau and Jeremy Sauget "Firmware and bootloader" *article,2012*

Acknowledgments

I am thankful to all my teachers, friends, well wishers and to all those many who have helped me both directly and indirectly in this project. I want to specially thank my advisers for guiding and showing the path through the hurdles I faced. My colleague Mr. Prasann Chaphe always has provided me support technically and morally throughout the project for which I am grateful to him. Also, I take this opportunity to thank Mr. Pratik Bhalerao and Mr. Saurav Kumar Behera (employees of ANTfarm Robotics Pvt. Ltd.) without whom it would not be possible to proceed ahead in this project.

Date: _____ Saurabh Kumar Panda