Implementation of Application Layer Protocol for an Active RFID System

Ambuj Agrawal

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Electrical Engineering

Majid Manteghi, Chair

R. Michael Buehrer

Patrick R. Schaumont

2nd August 2011

Blacksburg, VA

Keywords: RFID, ATmega328P, CC2520, CC2591, accelerometer, TagSense Inc.

Implementation of Application Layer Protocol for an Active RFID System

Ambuj Agrawal

Abstract

The emerging technology of active RFID tags has strong potential in the areas of real time health monitoring, sorting of cargo, and large scale inventory management because of their longer communication range and larger data storage capacity. The market of active RFID is growing very rapidly and therefore there has been an increase in the number of companies engaging in this field. But very often it is found that the products available in the market are not always suited to the application at hand. To overcome this problem, off the shelf active RFID products which were reconfigurable and followed a standard PHY and MAC layer protocol were used for this work. By reprogramming the application layer protocol of the RFID hardware, these devices were made suitable for the desired application. This also allowed the RFID tags to extend their functionality by interfacing extra modules with themselves. The work presented in this thesis describes the way in which the microcontroller on board the active RFID tags and readers can be programmed so that the functionality of the RFID hardware can be changed as per requirements. It also shows that extra modules can be added to the tag by successfully interfacing an accelerometer module with the tag.

Acknowledgement

This thesis has only been possible because of the help that I have received from many people. First and foremost, I would like to thank Dr. Majid Manteghi and Ashwin Amanna for sponsoring my master's thesis project. I would like to thank Dr. Manteghi for his supervision throughout the project. He was the one who kept me motivated to complete this project in time. His honest opinions about my work inspired me to put in more effort and his accessible nature always encouraged me to reach him with any problem that I had.

I would like to thank Ashwin Amanna for the guidance that I received from him throughout my stay at Virginia Tech. All the conversations that I had with him were very meaningful and helped me a lot in my research and life in general.

I would also take this opportunity to sincerely thank Dr. R. Michael Buehrer for all the advice that I have received from him during the past one year. It was unfortunate that the project we were working on didn't amount to anything but still I cherish all the conversations that I had with him and thank him for taking out time for me to talk about research and academics.

I would also like to thank Dr. Patrick R. Schaumont for agreeing to be on my committee. I asked him to be on my committee at a very late moment and it was very gracious of him to accept my request. I would also like to thank him for providing suggestions regarding my thesis.

J.Randall Nealy always helped me out with all the hardware problems I faced and for that I am very thankful to him. All the testing performed in this thesis would not have been possible without him.

Overall I would like to say thank you to my friends in the Mobile Portable Research Group (MPRG) lab and the Virginia Tech Antenna Group (VTAG) lab for going through my thesis and offering suggestions.

Finally, I would like to express my gratitude towards my family for their wonderful support during my graduate studies.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The motivation for this work was the current requirements of the railway industry. Railways have used RFID tags for many years and its use has proved beneficial in inventory control. But still this technology is underutilized mainly because of the limitations of passive RFID. An active RFID system for railway vehicle identification and position has been designed and developed [1] but this technology has not been exploited to its full potential by the railways. The railway industry's need for active RFID can be illustrated by the following two requirements-

1) A system that can keep track of the cargo carried on top of a rail truck.
2) A system that can keep track of the mileage of the rail car.

For both of these systems passive RFID technology will not be favorable because of its limited communication range and data storing capability. Active RFID can be a better alternative for implementing the abovementioned systems because it does not have the same drawbacks as the passive RFID technology.

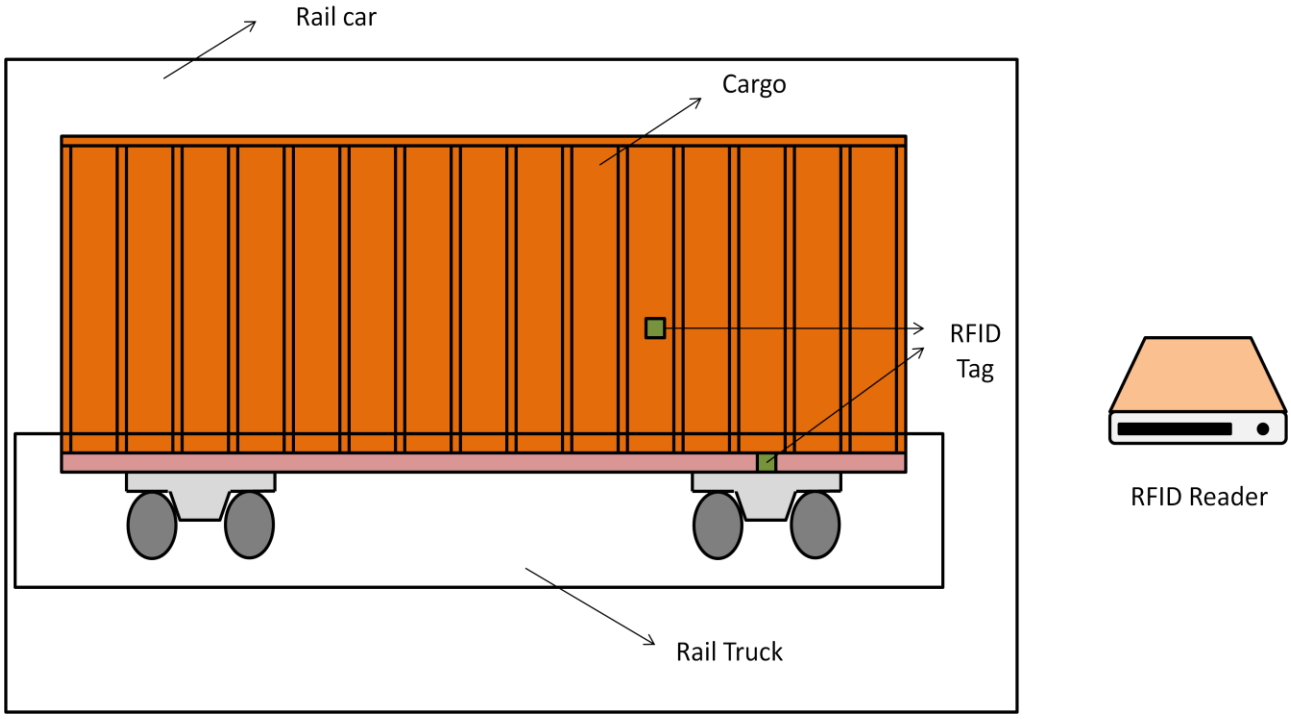


Figure 1.1: A system that can keep track of the cargo carried on top of a rail truck

As shown in fig. 1.1, for the first system we can fix one RFID tag each on the cargo and the rail truck. These tags can either communicate directly with each other or the tag on the cargo can send its data to a reader and then the reader can send the data to the other tag on the rail truck.

The tags are mostly in the sleep state and wake-up for a very small duration and then again go back to the sleep mode. If both the tags have to communicate directly then both the tags have to wake-up at the same time which will be difficult to achieve. Therefore the second alternative will be better when a system with low complexity is desired. Either way the tag on the rail truck will be able to maintain a record of all the cargo that is being carried on top of it. If there is some way to know the weight of the cargo then that information can also be passed on to the tag on the rail truck. Thus even a system as simple as this can be used to record lot of useful data related to the rail truck by using active RFID technology.

For the second system, we can keep track of the waypoints that the railcar passes. This way at the last station the distance travelled by the rail car can be calculated depending on the source station and the waypoints passed by the railcar. This is illustrated in fig. 1.2. For this system, passive RFID cannot be used because of the speed at which the railcar crosses the waypoints.

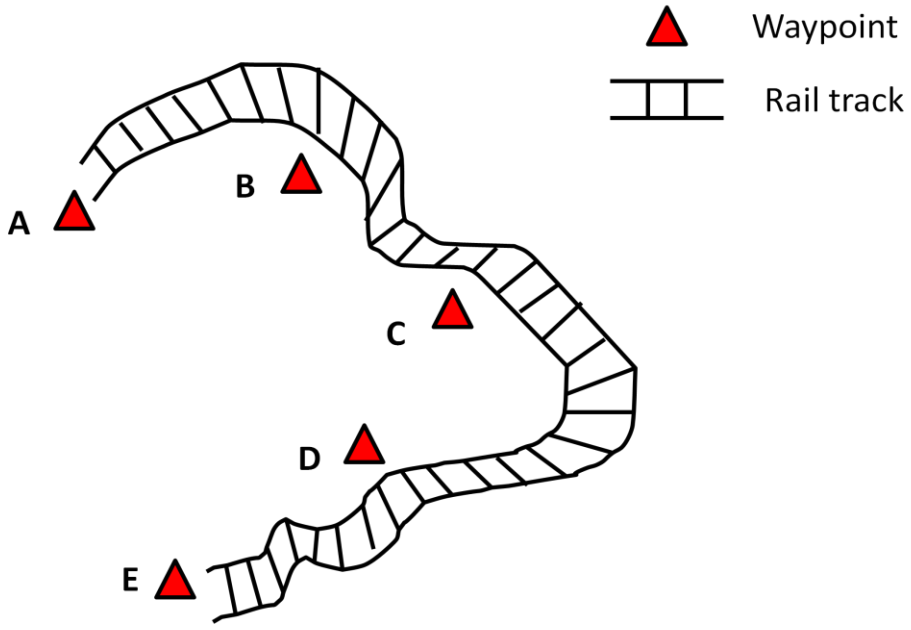


Figure 1.2: A system that can keep track of the mileage of the rail car

Because of the time constraints, an off the shelf tag was needed that could cater to the various needs of the railways. But after the market survey it was found that no single tag will fulfill all the necessities. Many of the RFID companies implement proprietary protocols for communication between the reader and the tag but some of them also implement standard protocols like IEEE 802.15.4. Many active RFID tags also contain a microcontroller for storing and processing data which can be reprogrammed. Therefore it was decided to buy a tag which followed a standard protocol so that only the application layer of the tag needed to be reprogrammed.

In the future there might also be the need to add GPS module or an accelerometer module to the tag. Also some sensors might be required to be integrated with the tag for the health monitoring of the rail car. As mentioned already, off the shelf tags do not contain all these capabilities. If there are unused input/output pins present on the tag then by reprogramming the tag all these extra functionalities can be added to it.

The tag used for this project is from *TagSense Inc.* This tag was used to provide a general purpose RFID platform which can be configured over the air. This objective was achieved by programming the tag with an application layer protocol which was similar to that of *TagSense Inc.* with some changes. The problem with the tag from *TagSense* was that in its present form it lacked the capability of interfacing additional hardware like a GPS module or an accelerometer module directly with itself. By reprogramming the tag, extra functionality can be added to the tag. In this project an accelerometer module was successfully integrated with the tag. In the future the tag can again be reprogrammed as and when needed to add more functionality.

1.2 Problem Statement

The previous sub-section explained the reasons for using active RFID technology in this work and also briefly explained the objective of this thesis. In this sub-section the problem being tackled in this thesis will be explained in more detail.

The commercial active RFID products available in the market do not always meet all the requirements. To solve this problem, there is a need to take one of these off the shelf products and modify them as per the needs of the application. Active RFID tags consist of three main components-a microcontroller, a RF transceiver and a RF front end. The microcontroller controls the functionality of the whole tag. Thus by reprogramming the microcontroller inside the tag the functionality of the tag can be modified as required.

The RF transceiver takes care of the physical layer protocol. It is responsible for sending and receiving data over the air. It also takes care of the medium access control method to allow multiple tags to talk to the reader at the same time. The RF front-end houses a power amplifier and a low noise amplifier to extend the communication range of the tag. The microcontroller implements the application layer protocol. The application layer protocol defines the frame format of the commands that can be sent from the reader to the tag and also the format of the different data packets that can be transmitted from the tag.

The data frame to be transmitted by the tag is prepared by the microcontroller and sent to the transceiver, the transceiver sends it to the RF front-end and from the RF front-end the frame is transmitted over the air. While receiving a command from the reader the reverse operation happens. The application layer protocol is not only responsible for preparing the data frames to be transmitted, it also parses the received frame and takes the appropriate action. It is also in charge of dealing with received data frames which contain transmission errors.

In this thesis both the active RFID tag and the reader are reprogrammed. This work only deals with the application layer protocol. The physical layer protocol and the medium access control protocol cannot be changed by reprogramming the microcontroller.

One of the major reasons for reprogramming the active RFID devices in this thesis was to integrate additional modules with the tag. The accelerometer module was successfully interfaced with the tag in this work. The microcontroller inside the tag contains a digital SPI interface. This same interface is also present in the accelerometer module. Through this interface the microcontroller can communicate with the accelerometer. The microcontroller also contains an USART interface. This module can also be used to communicate with the outside world.

Emphasis is also paid on the power consumed by the tag. Most of the power is used when the tag is transmitting or receiving data. Therefore, the tag remains in the sleep mode for majority of the time to save power. During this time it can shut down all its main components. In fact the microcontroller provides the option to choose from six programmable sleep modes. It is shown in the results section that the reprogrammed tag without the accelerometer module (to have a fair comparison) uses as much power as the original tag. This shows that the power efficiency of the tag was not affected by reprogramming it.

The length of the transmitted packet from the tag is kept as small as possible to reduce the power consumption of the tag. This also helps in increasing the number of tags that can talk simultaneously to the reader.

A laptop is used to control the active RFID system implemented in this thesis. The reader is connected to the laptop. A GUI is also implemented to send commands to the reader and the tag. The GUI also displays the data received from the tags by the reader.

1.3 Background Information

1.3.1 Active RFID v/s Passive RFID

Passive RFID tags do not have a battery power source and thus they rely on the energy provided by the RFID reader to communicate. There are two ways in which the passive RFID tag can respond to the signal from the reader [2]-

1) Reflect energy from the reader.
2) Absorb a part of the energy from the signal to generate its own quick response.

The passive RFID tag stores the energy of the signal from the reader in an on board capacitor. This capacitor, when has enough charge built in it, is used as the power source by the tag to transmit its own modulated signal [3].

The advantages of passive RFID are –

1) They do not require battery to function and therefore have a very long lifetime.
2) They are very cheap to manufacture.
3) They are smaller in size.

The disadvantages of passive RFID are-

1) They have a short communication range.
2) They have a very small data storage capacity.
3) They have limited multi-tag collection capabilities.
4) They can record data only when they are in the vicinity of a reader.

The applications of passive RFID are -

- Inventory control
- Electronic article surveillance
- Asset tracking
- Animal and vehicle identification
- Library management systems

In general passive RFIDs can be used where large amount of data storage is not required and where there is constrained asset movement [4].

Active RFID tags on the other hand have an internal power source in the form of a battery within the tag that continuously powers the tag. In addition to the on board power supply they also contain on board electronics. The on board electronics may include microprocessors, sensors and input/output ports [5]. Because of the electronics they can find use in many different applications.

The advantages of active RFID are –

1) They provide a larger communication range.
2) They provide larger read/write data storage capacity with sophisticated data manipulation and search capabilities available [2].
3) They may carry sensors on them which can be used to collect some specific information.
4) They have the ability to continuously monitor and record data from the sensors.
5) Active RFID technology can be used to collect information from multiple tags moving at high speed.
6) The tags do not require line of sight.

The disadvantages of active RFID are-

1) The tag's lifetime is limited because it cannot function without a battery.
2) They are more expensive than passive RFIDs.

3) The tags are larger in size.
4) There may be expensive misreads because of battery outages in an active tag [6].
5) There is no commonly agreed set of protocols for active RFID. The majority of active RFID protocols are proprietary.
6) They require complex receivers and anti-collision type software [7].

	Active RFID	**Passive RFID**
Tag Battery	Yes	No
Availability of tag power	Continuous	Only within the field of a reader
Required Signal strength from Reader to Tag	Low	High
Communication Range	Long range (from a few meters to 100 meters)	Short range (from a few centimeters to 5 meters)
Multi-Tag Collection	Can collect information from multiple tags moving at high speed	Limited multi-tag collection capability
Sensor Capability	Ability to continuously monitor and record data from sensors	Can collect and transfer data from sensors only when in range of a reader
Data Storage	Large read/write data storage with sophisticated data search and access capabilities available	Small read/write data storage
Frequency of Operation	Operate at higher frequencies-433 MHz, 2.4 GHz and 5.8 GHz	Operate at lower frequencies-128 kHz, 13.56 MHz,860-960 MHz
Cost	Expensive-tags cost from 20$ to 100$	Cheap-cost of tags is typically in cents
Lifetime	Limited by the lifetime of battery	Unlimited
Size	Large	Small
Protocols	Majority of them are proprietary	Standardized

Table 1.1: Summary Of Differences Between Active And Passive RFID

The applications of active RFID are-

- Vehicle Identification
- Asset Tracking
- Real time location systems (RTLS)
- Remote sensor monitoring
- Inventory control

The primary industries using active RFID are automotive, transportation, logistics, healthcare and military [8].Table 1.1 summarizes the major differences between active and passive RFID.

1.3.2 Active RFID System

The active RFID system mainly consists of three components-tag, reader and server as shown in fig.1.3. The tag communicates with the reader in accordance with a particular protocol which can be either a standard protocol or a proprietary one. This protocol will also specify the anti-collision algorithm to be followed to keep the collisions between data packets from different tags to a minimum. A collision occurs when two or more tags transmit a packet at the same time. The anti-collision protocol allows the reader to collect information from different tags simultaneously thus increasing the throughput of the system.

Once the data is collected by the reader, it has to be passed on to a server. This can be done using USB port, serial port, Ethernet cable or through wireless communication. The server runs advance algorithms and processes the data to display useful information in human readable format.

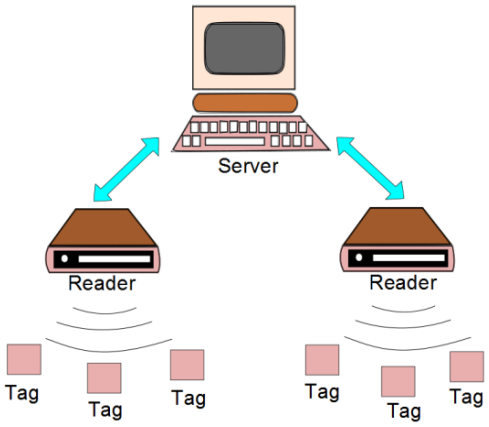


Figure 1.3: Active RFID System [9]

One important thing to note here is that the tags never talk among themselves directly. The system is similar to a master slave system where the reader is the master and the tags are the slaves. The tags will always talk to the reader first and if required the reader will then pass on the information from one tag to another.

1.3.3 Wireless Sensor Network v/s Active RFID System

A Wireless Sensor Network (WSN) consists of spatially distributed autonomous sensors. These sensors monitor physical or environmental parameters like temperature, humidity, pressure, etc. and cooperate with each other to pass the data through the network to one main location [10].

A Wireless Sensor Network is composed of multiple nodes. These nodes contain sensors inside them for measuring a physical or an environmental condition. In addition to sensors, these nodes also have a microcontroller, transceiver, external memory and a power source.

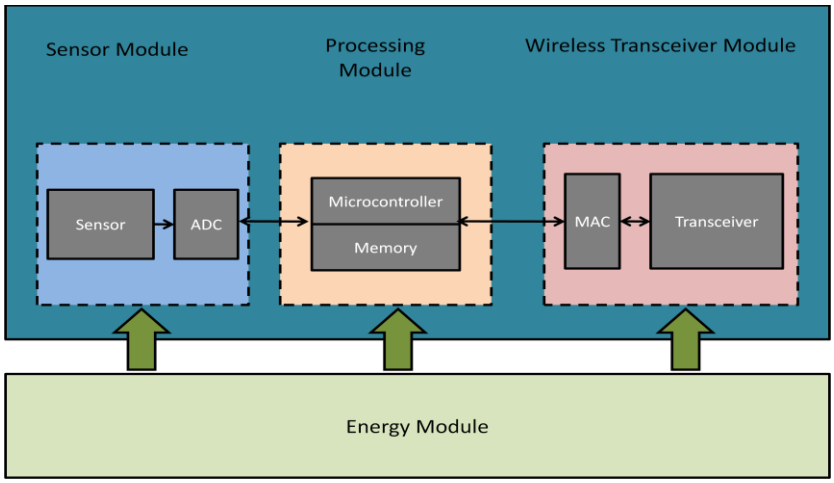


Figure 1.4: Architecture of a typical sensor node [11]

The architecture of a typical sensor node is shown in fig. 1.4. The analog to digital converter (ADC) is can be present within the sensor or the microcontroller and is not present as a separate block. Similarly the medium access control (MAC) block is can be present within the transceiver module. As will be shown later, this architecture is very similar to the architecture of an active RFID tag that has been used in this project. As a matter of fact the difference between an active RFID tag containing sensors and a wireless sensor node is rather small [12].

The main difference between WSN and an active RFID system is the way in which information is transferred in these two systems. In WSN data reaches a base station after travelling through several sensor nodes. As the sensor nodes have the ability to cooperate with each other, they have no problem in transferring data among themselves. Therefore communication within a WSN is multi-hop. As already mentioned, in an active RFID system a tag cannot directly transfer its data to another tag. The data has to pass through the reader before passing onto another tag. That is to say, the communication within an active RFID system is single-hop.

In addition to the nature in which the information moves in the two systems, there are other differences as well. In WSN the nodes are usually static while in an active RFID system the tags are attached to moving objects. Also the protocols followed in the two systems may differ [13].

Therefore the difference between a wireless sensor node and an active RFID tag enabled with sensors does not lie in their respective hardware but in the cooperative capabilities of these devices. The hardware used for this project can qualify either to be a wireless sensor node or an active RFID tag. Henceforth, it will be referred to as an active RFID device and not as a wireless sensor node as the implemented system only permits single-hop communication.

WSN implements multi-hop for power reasons. This can be explained better by calculating the path loss when a single hop is used for communication as compared to the path loss when multiple hops are used. Path loss is the ratio of transmit power to receive power, assuming isotropic antennas. For free space:

$$L_p = \frac{P_T}{P_R} = \left(\frac{4\pi R}{\lambda} \right)^2 \qquad (1.1)$$

where L_p is the path loss, P_T is the transmitted power, P_R is the received power, R is the distance between the transmitter and the receiver and λ is the wavelength of the RF signal used for communication [14].

For practical purposes equation (1.1) will not hold true. A very simple but useful model is obtained as follows-

For distances less or equal to a reference distance (which is typically equal to 1λ) free space path loss model is used:

$$L_{p,ref} = \frac{P_T}{P_R} = \left(\frac{4\pi R_{ref}}{\lambda} \right)^2 \qquad (1.2)$$

For distances greater than the reference distance:

$$L_p = L_{p,ref} \left(\frac{R}{R_{ref}} \right)^{\alpha} \qquad (1.3)$$

where α is the path loss exponent and typically its value can lie anywhere between 2 and 5 [14].

Therefore from equation (1.3) we can conclude that when single hop communication is used:

$$L_{p,1-hop} \propto R^{\alpha} \qquad (1.4)$$

The path loss when N hops are used for the same transmitter-receiver distance (assuming distance is equally divided between the N hops):

$$L_{p,N-hops} \propto N \bullet \left(\frac{R}{N} \right)^{\alpha} \qquad (1.5)$$

From equation (1.4) and equation (1.5) the relative transmit power needed for the same received power in the case of multi-hop communication as compared to single-hop communication can be shown to be:

$$\frac{P_{T,N-hops}}{P_{T,1-hop}} \propto \frac{1}{N^{\alpha-1}} \qquad (1.6)$$

Equation (1.6) shows that by using multi-hop communication there can be a significant reduction in the power consumed by the sensor node. But it clearly increases the complexity of the system.

For two nodes to communicate with each other, one of them has to be in the receiving mode and the other in the transmit mode. This is not very easy to achieve when we consider the power constraints faced by the nodes. The nodes have to follow complex MAC protocols in order to transfer data in an energy-efficient and reliable manner.

Although it may seem that WSN is more power efficient than an active RFID system but it is not entirely true. In many RFID systems the tag does not need to transmit or receive very often which greatly reduces the power consumption. Let us consider the example of the protocol implemented by *TagSense Inc*. They call the protocol as *Tag Talks First* which means that the communication between the reader and the tag is always initiated by the tag. Most of the power consumed by the tag is when it's either transmitting or receiving. Because the tag does not have to listen to the reader periodically, a considerable amount of power is saved. The tag can also be configured over the air by the reader to transmit very infrequently like once every hour or when an event like temperature crossing a particular threshold occurs. The tag stays in the receive mode for a few millisecond after transmitting to receive commands from the reader. Therefore by minimizing the time period during which the RF functions of the tag are used, the battery lifetime can be increased.

1.4 Organization of Thesis

The thesis is divided into six major chapters. Chapter two gives an overview of the system architecture and *TagSense* protocol. Chapter three of the thesis describes the hardware components present on board the tag and the accelerometer module interfaced with the tag. Main hardware components present on the tag are Atmel's ATmega328P microcontroller, Texas Instruments' RF transceiver CC2520 and Texas Instruments' RF front-end CC2591. Chapter four describes the development tools used to write, debug and compile the microcontroller code and also discusses the various libraries used. It also explains algorithms used for implementing the application layer protocol. Chapter five discusses the software used for writing the Graphical User Interface (GUI). This chapter also explains how the GUI can be used to configure the tag. Chapter six shows all the testing performed with the tag and the results obtained. Chapter seven suggests possible future enhancements to the project and concludes the thesis.

2 System Architecture and Protocol Overview

2.1 System Architecture

The system developed in this thesis is similar to the one described in section 1.2.2. The laptop acts as a server, the laptop is connected to the active RFID reader through a serial connection (RS-232 protocol) and the active tags communicate with the reader over the wireless channel.

The ZT-500 tag from *TagSense Inc.* is used as the hardware for both the active RFID reader and the active RFID tag. Although the hardware used for both the tag and the reader is same, the program loaded in the microcontroller present inside the ZT-500 tag is different. The hardware used in this work is explained in more detail in section 3.

The microcontroller present in ZT-500 tag is ATmega328P from Atmel. This microcontroller was programmed using AVR Studio 4. To display the data received from the tags and to send commands to the tag a Graphical User Interface (GUI) was developed using Perl language and Apache web server.

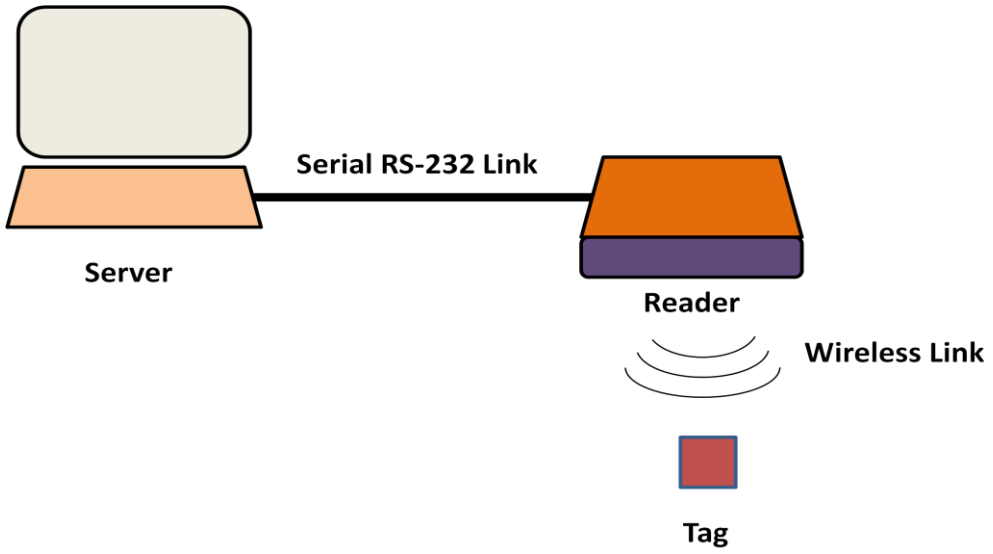


Figure 2.1: Communication links in the implemented active RFID system

There are basically two communication links that exist in the system as shown in fig. 2.1- a) the serial RS-232 link between the reader and the laptop and b) The wireless link between the tags and the reader. The following parameters are used with the serial link that exists between the reader and the laptop-

- 8 data bits
- 1 stop bit
- No parity bits

- No flow control
- Baud rate-38400 bps

The tags and reader communicate over the 2.4 GHz ISM band. These devices follow the IEEE 802.15.4 protocol. Therefore they use Direct Sequence Spread Spectrum (DSSS) for communication and provide a data rate of 250kbps. The O-QPSK modulation is used in 802.15.4 protocol when operating in the 2.4 GHz band [15].

The transmit power of the tags and reader is programmable and can go up to 17 dBm. The receiver sensitivity is around -98 dBm [16]. The communication range of these devices can go up to around 1700 meters [17].

The tag and the reader consist of three main components- a microcontroller, a RF transceiver and a RF front-end. The microcontroller controls the functionality of the transceiver and the RF front-end. The microcontroller also helps the reader to communicate with the laptop through the USART interface. The SPI interface in the microcontroller is used to communicate with the RF transceiver and the external accelerometer that has been integrated with the tag.

The data is received from the tags by the reader over the wireless channel and passed onto the laptop through the USART interface. This data is then displayed on the GUI. The reverse happens when a command is sent from the laptop. The command reaches the reader through the USART interface, the reader parses the command and if the command is not meant for it then it stores it in its memory. This command is delivered to the tag whenever the tag contacts the reader. If the command was meant for the reader then the reader takes the appropriate action. The protocol will be explained further in the next sub-section.

The features of the microcontroller that have been programmed in this thesis and their purpose is as follows-

1) The three types of memories present inside the microcontroller- SRAM, EEPROM and Flash, are used to store different kinds of data.
2) General digital input/output ports are programmed as necessary for a variety of purposes (some pins are used as digital inputs, some as digital outputs and some as inputs to the ADC module inside the microcontroller).
3) Timer/Counter2 module in the microcontroller is used to implement a real time clock on the tag and the reader.
4) The Serial Peripheral Interface (SPI) is used to communicate with the RF transceiver and the accelerometer. The microcontroller is always configured as master in this thesis whenever using this module.
5) The USART interface in the reader is used to communicate with the laptop. The serial RS-232 protocol is used when communicating with the laptop. As the output voltage levels of the USART module are not compatible with the RS-232 protocol, an external adaptor is used to convert the USART output to the appropriate voltage level.

6) One of the six programmable sleep modes provided by the microcontroller is used to save power.
7) The ADC is programmed to convert the analog signal from the temperature sensor into a digital signal.
8) The functionality of the RF transceiver is controlled by the microcontroller through the SPI interface by programming its various registers.
9) The microcontroller also controls the receive mode gain of the RF front-end.

Section 3 and section 4 explain in more detail the way in which different parts of the tag and reader are programmed.

2.2 TagSense Protocol

The *TagSense* protocol specifies the set of rules followed by a host computer to communicate with *TagSense's* active RFID tags through the active RFID reader. In this protocol all communication is initiated by the tag and not controlled by the reader. The host computer sends the command to the reader, which stores it in its memory. Whenever the reader receives a packet from the tag, it transmits the first command in its memory meant for the tag with the corresponding tag ID. The tag after transmitting a packet and before going back to the sleep mode goes into the receive mode for some time to receive any command from the reader. This way the tag saves battery as it does not have to periodically wake up to listen to the channel for commands from the reader [18].

Figure 2.2: Tag command string sent by host in the TagSense protocol [18]

There are two types of commands that can be sent from the host- the reader commands and the tag commands. The tag commands contain unique command handles and have two parts as shown in fig. 2.2. The first part contains the reader envelope information which contains instructions for the reader. The instructions tell the reader how a command is to be delivered to the tag, the command queue operation and special commands like synchronizing its time with the tag. The second part contains the tag command which is not parsed by the reader and delivered as it is to the tag [18].

The host can command the reader to perform the following key actions [19]-

1. Return the command handles for all tag commands stored in the queue.
2. Return the number of commands in the queue.
3. Delete a command.
4. Return the command associated with a specific command handle.
5. Flush the command queue.

6. Set reader time.
7. Return reader time.
8. Append reader time to end of every tag packet that is received and forwarded to the host.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Halt Mask	Transmit on Interrupt Mask bit	TX Interval Setting Mask Bit	TX Interval Report Mask Bit	TX Power Setting Mask Bit	TX Power Reporting Mask Bit	Tag ID Mask Bit 0	Network Control Field Extension Mask Bit

Table 2.1: TagSense Protocol-Network Field Mask Byte [15]

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable /Disable Tag Halt	Enable /Disable Transmit on Interrupt	Set/Read TX Interval Bit	Enable /Disable TX Interval Report	Set/Read TX Power Bit	Enable /Disable TX Power Report Bit	Set/Read Tag ID Bit	Network Control Field Extension Bit

Table 2.2: TagSense Protocol-Network Field Action Byte [18]

The tag command sent to the tag contains a *command header* and a *packet control header*. The *command header* specifies whether the tag has to send an acknowledgement packet back to the reader and the particular mode whose parameters will be affected by the command. There are up to 7 user programmable modes, a default mode and a data dump mode. At a given time the tag can be in any one of the nine modes mentioned above. Different configuration parameters can be assigned to different modes [18]. For example in *Mode 1* the transmit interval of the tag can be set to ten seconds while in *Mode 2* it can be set to one minute. The *packet control header* specifies the fields present in the command packet. The following fields can be present in a tag command packet [18]-

- *Network Field:* This field handles network related commands, e.g., setting the transmit interval, transmit power, etc.
- *GPIO Field:* It handles GPIO related commands, e.g., setting the pins to be digital output or digital input, output of a digital pin to be logical zero or logical one, etc.
- *Alarm Field:* It handles alarm related commands, e.g., tag should enter *Mode 3* at 2 PM, *Mode 4* should be entered at 10 PM, etc.
- *Trigger Field:* It handles trigger related commands, e.g., the tag should enter *Mode 3* if the temperature is above 90^{o}F, etc.
- *User Memory Field:* The EEPROM memory in the tag can be used to store user data. This field can be used to command the tag to read or write data to the EEPROM.
- *Data Sampling Field:* The flash memory is used to store data records by the tag. The data records are used to store the sensor data. This field can be used to command the tag to start recording data or to read the data from the flash memory.

- *RTLS Field:* This field handles data relating to Real Time Location System functions. This field is currently not implemented in the protocol.

The field bytes are nominally 2 bytes in length. The first byte is the mask byte and the second byte is the action byte. The mask byte indicates the parameters to be configured while the action byte specifies how a particular parameter has to be modified [18]. For example, the network field mask byte and action byte are shown in table 2.1 and table 2.2 respectively.

The receiver is always in the receive mode (except when it goes briefly into the transmit mode to transmit commands to the tags) to receive data from the tags. Whenever the tag wants to communicate with the reader, it first senses the channel and if the channel is clear then it transmits its data packet. In this way the communication link is always established by the tag. The reader can then transmit a command back to the tag if needed.

There can be three types of data packets that can be transmitted by the tag- the beacon packet, the response packet and the data dump packet. On receiving a packet, the RF transceiver module in the reader performs the following operations [15]-

1. Detection and removal of PHY synchronization header
2. Reception of number of bytes specified by the frame length field
3. Automatic frame check sequence (FCS) checking

The RF transceiver will not discard the packet even if the FCS is not correct. It will store the received frame minus the last two FCS bytes in the RX FIFO buffer. In place of the FCS bytes, two other bytes are stored which contain useful information like the RSSI value, whether the FCS is correct or not, etc. Thus whenever a frame with incorrect FCS is received, it is discarded. To make the reception of frames more foolproof, the format of the received frame is matched against the format of the three types of packets transmitted by the tag and if no match is found then the frame is discarded.

The CSMA-CA protocol is followed by the RF transceiver to allow multiple tags to talk simultaneously to the reader. When packets from several different tags are received by the reader at the same time, it distinguishes among them by looking at the two byte tag ID present in the data packet.

2.3 Research Issues

The main research issues related with this work are –

1) The power consumption inside the tags (as their lifetime is limited by the battery life).
2) Handling of multiple tags in the system.

The *TagSense Inc.* protocol is an asynchronous protocol and thus can provide longer battery lifetime as the tag does not have to wake up periodically to check whether the reader intends to transfer data to it or not.

To handle more number of tags, two factors have to be considered- the transmission interval and the transmission duration. Therefore, the maximum number of tags that can talk simultaneously to the reader will be-

$$\frac{Tx_{INT}}{Tx_{DUR}} \qquad (2.1)$$

where Tx_{INT} is the interval between two back to back transmissions by a tag and Tx_{DUR} is the time it takes to transmit a packet by the tag.

Reducing the size of the packet transmitted by the tag will result in the following-

1) The power consumption inside the tag will reduce as the time for which the tag's transmitter is switched on will decrease.
2) From equation (2.1), the number of tags that can talk to the reader simultaneously will increase.

The *TagSense Inc.* protocol does not specify the length of the packet to be transmitted by the tag. In this thesis, the tags only transmit the most necessary information in a packet so that the transmit duration can be kept to a minimum. The tags will transmit additional information only if asked explicitly by the reader. Thus by reducing the transmit duration in this thesis the power consumption and multiple tag handling capacity of the implemented system were improved.

3 Hardware Description

The following hardware was used in this project-

1. Three ZT-500 Active RFID tags from *TagSense Inc.*(one is used as a reader while the others are used as tags)
2. RS-232 Adapter
3. USB to Serial converter
4. AVR mkII Programmer
5. *Bosch Sensortec's* BMA150 Accelerometer

3.1 ZT-500 Active RFID Tag

The ZT-500 tag from *TagSense Inc.* is a reprogrammable tag and can be configured over the air in real time by the user. The ZT-500 tag contains the following major components-

1. ATmega328P AVR microcontroller from *Atmel*
2. CC2520 2.4 GHz IEEE 802.15.4/ZigBee compliant RF transceiver from *Texas Instruments*
3. CC2591 2.4 GHz RF front-end from *Texas Instruments*
4. 32.768 kHz Crystal Oscillator
5. Temperature Sensor
6. Energy Module

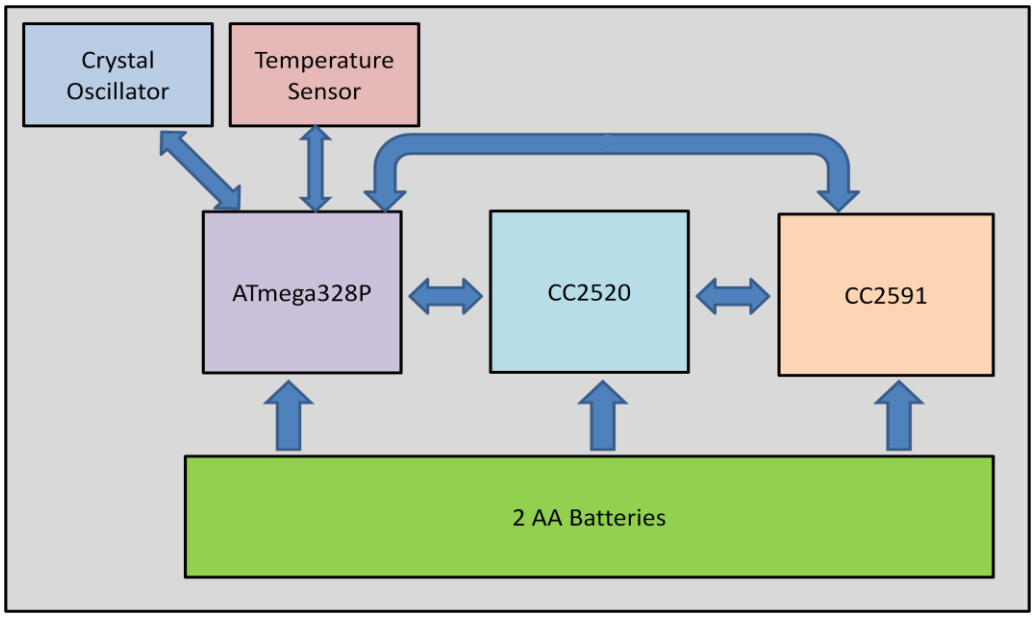


Figure 3.1: Layout of the ZT-500 Tag from TagSense Inc.

The layout of the tag is shown below in fig. 3.1. The AVR ATmega328P microcontroller controls the functions of the other components of the tag. ATmega328P has a built in ADC and the data logging in the tag also takes place inside it. The crystal oscillator is used to implement a real time clock while the temperature sensor is used to monitor the temperature of the tag. The CC2520 transceiver module is used to transmit and receive data over the air. The CC2591 module provides the RF front-end and contains a power amplifier and a low noise amplifier inside it. Two AA batteries are used to provide power to the tag.

3.1.1 ATmega328P Microcontroller

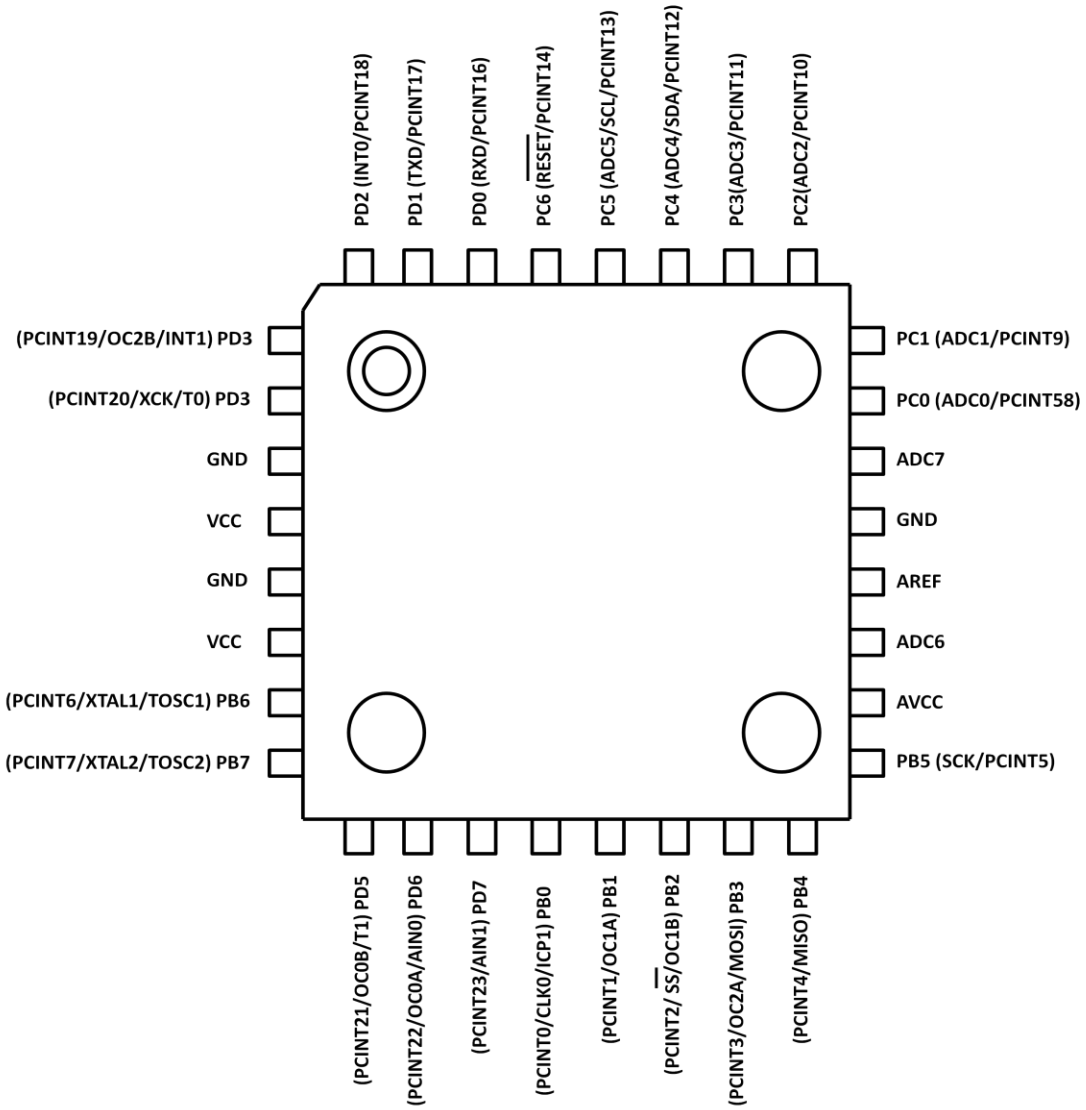


Figure 3.2: ATmega328P Pinout [20]

The ATmega328P microcontroller is a high performance, low power 8-bit microcontroller with advanced Reduced Instruction Set Computing (RISC) architecture [20]. The pin configuration of ATmega328P is shown in figure 3.2. The operating temperature range is from -40 ºC to 85 ºC and the operating voltage range is from 1.8V to 5.5V.

The main features of ATmega328P that have been used in this project are-

- *SRAM Data Memory:* ATmega328P contains 2K bytes of Static Random-Access Memory (SRAM). It's a volatile memory which means that data stored inside it is lost when the power to the memory is removed. This memory is used primarily to store the stack contents and the variables created during runtime.

- *EEPROM:* EEPROM stands for Electrically Erasable Programmable Read-Only Memory. It's a non-volatile memory and therefore can retain the data stored in it even in the absence of power. The ATmega328P microcontroller contains 1K byte of EEPROM. This memory should be used to store sparingly changing data such as device parameters. For example the TAG ID can be stored in the EEPROM. It's not a good idea to store variable data in this memory because it has a limited number of write cycles. The EEPROM in ATmega328P can endure at least 100,000 write/erase cycles. There is no restriction on the number of read cycles.

- *Flash Memory:* The Flash memory is used to store the program. ATmega328P has 32K bytes of flash memory. The Flash memory in this microcontroller can endure 10,000 write/erase cycles. This memory is divided into two sections – the application flash section and the boot section. The application flash section is used to store the application code. The boot section can be used to store data or boot loader software. Boot loader can communicate with the outside world via the pins of the microcontroller and can also change the program in the application flash section. The size of the boot section is configured with the help of fuses. The flash memory is written in a page-by-page fashion. This means that we cannot just write one byte of data to the memory. Whenever a write operation is performed an entire page of data is written to the flash memory. For ATmega328P the page size is 64 bytes.

- *Input/Output Ports:* ATmega328P contains general digital I/O ports. There are three I/O ports- Port B, Port C and Port D. Port B and Port C contain eight pins while Port D contains seven pins. Three registers are assigned for each port-Data Register: *PORTx*, Data Direction Register: *DDRx* and Port Input Pins register: *PINx* (where *x* denotes the port letter). *DDRx* and *PORTx* are read/write while *PINx* is read only. But if logic one is written to a bit in the *PINx* register then it will cause the corresponding bit in the *PORTx* register to toggle. *DDRx* register is used to configure the direction of the port pins. If logic one is written to *DDRxn* (where *n* denotes the pin number) then it's configured to be output otherwise it's configured to be input. When pin is configured to be output and logic one is written to *PORTxn* then the corresponding pin will be driven high otherwise it will be driven low. When pin is configured

to be input and logic one is written to *PORTxn* then the pull-up resistor for that pin is activated otherwise the pin will be set to tri-state. If the *Pull-Up Disable-PUD* bit in *Micro Controller Unit Control Register (MCUCR)* is set then pull-up function will be disabled for all pins in all ports. Table 3.1 summarizes the different port pin configurations.

DDRxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pin will source current if externally pulled low
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Table 3.1: Port Pin Configurations [20]

Independent of the direction of the pin, the value of the pin can be read through the *PINxn* register bit.

- *Clock Systems:* The clock systems present in ATmega328P are-
  1. CPU Clock-clk_{CPU}: Provides clock to modules concerned with core CPU operations like General Purpose Register File, Status Register and SRAM.
  2. I/O Clock-$clk_{I/O}$: This clock is used by majority of I/O modules like Timer/Counters, Serial Peripheral Interface (SPI) and Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART).
  3. Flash Clock-clk_{FLASH}: Provides clock to the flash and EEPROM memory.
  4. Asynchronous Timer Clock-clk_{ASY}: Allows the Timer/Counter module to function asynchronously with the help of an external clock.
  5. ADC Clock-clk_{ADC}: Provides clock to the ADC.

ATmega328P has the following clock source options which can be selected by flash fuse bits. Table 3.2 lists all the clocking options available and the corresponding setting of the fuse bits to select them. The clock source used in this project is the *Calibrated Internal RC Oscillator* with clock frequency equal to 8.0 MHz.

Device Clocking Option	CKSEL Fuse [3:0]
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128 kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Table 3.2: Device Clocking Options Select [20]

- *Sleep Modes:* ATmega328P provides various sleep modes to shut down unused modules, thereby saving power. The sleep modes provided by this microcontroller are as follows-
  1. Idle Mode: Halts clk_{CPU} and clk_{FLASH}.
  2. ADC Noise Reduction Mode: Halts clk_{CPU}, clk_{FLASH} and $clk_{I/O}$.
  3. Power-down Mode: Halts all the clock systems.
  4. Power-save Mode: Similar to power-down mode but Timer/Counter2 keeps running.
  5. Standby Mode: This mode can only be used if an external crystal/resonator clock option is selected. It's also similar to power down mode but the oscillator is not stopped.
  6. Extended Standby Mode: This mode also can be used only when an external crystal/resonator clock option is selected. This mode is similar to power-save mode with the exception that the oscillator is kept running.

  ATmega328P wakes up from the sleep mode when an interrupt occurs. The interrupts that can be used as the wake-up source for ATmega328P depend on the sleep mode being used. The sleep mode used in this project is the power-save mode as it shuts down most of the modules but keeps the Timer/Counter2 running. This ensures that the real time clock functions properly even when ATmega328P is in the sleep mode.

- *Timer/Counter2:* Timer/Counter2 is an 8-bit module and has been used to implement a real time clock on the tag. It uses a 32.768 kHz external watch crystal as its clock source when it's working in the asynchronous mode. This clock can be scaled to a lower frequency by using the prescaler for this module. The counter increments by one after every clock cycle and can generate an interrupt when there is an overflow. To realize a real time clock the external crystal and the prescaler are used to generate an overflow interrupt every second. In this project the interrupt is generated every half a second.

- *Serial Peripheral Interface:* This module is used by AVR microcontrollers to communicate with the outside world. Most of the AVR programmers today use this module to transfer the program (hexadecimal code) from the laptop on to the AVR microcontroller. The data is transferred asynchronously between two devices using this module. One of the devices is configured to be the master while the other is used as the slave. The interconnection between master and slave is shown in fig. 3.3.

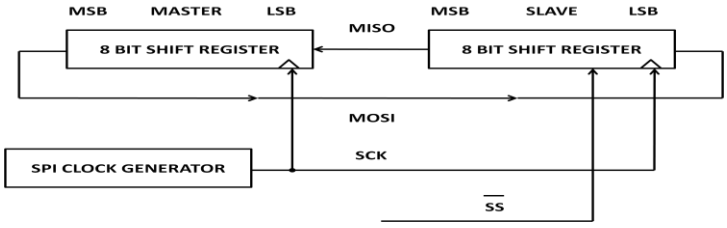


Figure 3.3: SPI Master-Slave Interconnection [20]

The wires used by this module are *MOSI, MISO, SCK* and *SS*. The system consists of a master clock generator and two 8-bit shift registers, one residing in the master and the other in the slave. The communication starts when the *Slave Select (SS)* pin of the slave is pulled low. The required clock pulses are generated on the *SCK* line. Data is always shifted from mater to slave on the *Master Out Slave In (MOSI)* line and from slave to master on the *Master In Slave Out (MISO)* line. The data is transferred after every clock pulse and therefore after eight clock pulses one byte of data is transferred from the master to the slave and from the slave to the master. The shift registers after eight clock pulses contain the new data instead of the data that was to be transferred. After the data transfer is complete, the *SS* line should be pulled high by the master.

The SPI has 4 different data modes depending on the phase and polarity of SCK with respect to serial data. The data mode being used is determined by the control bits CPHA and CPOL. Table 3.3 shows the different SPI data modes.

SPI Mode	Conditions
0	CPOL=0, CPHA=0
1	CPOL=0, CPHA=1
2	CPOL=1, CPHA=0
3	CPOL=1, CPHA=1

Table 3.3: SPI Data Modes [20]

- *Universal Synchronous and Asynchronous serial Receiver and Transmitter:* The USART is another module which allows the AVR microcontroller to communicate serially with other peripherals or devices connected to it. Before transmitting data, both the devices agree on a baud rate which is then used to sample the Rx and Tx lines at regular intervals. The USART clock is derived from the system clock. The *USART Baud Rate Register (UBRRn)* and the down counter connected to it provide the functionality of a programmable prescaler or a baud rate generator. *UBRRn* is a 16 bit register and is divided into two 8-bit registers-*UBRRnH* and *UBRRnL*. The counter counts down on every clock pulse and whenever it reaches zero or a write operation is performed on the *UBRRnL*, it's loaded with the value of *UBRRn*. Whenever the counter reaches zero a clock pulse is generated. Therefore the clock frequency generated by the baud rate generator is equal to $f_{OSC}/(UBRRn+1)$, where f_{OSC} is the system clock frequency. The transmitter divides the output from the baud rate generator by 2, 8 or 16 depending on the mode. The receiver uses the output from the baud rate generator directly. The value of *UBRRn* may not be an integer. This will lead to some errors in the data received or transmitted. The percentage error will depend on the system clock frequency and the transmit mode. The AVR datasheet should be consulted to choose a baud rate value which will have lesser percentage error. The error percentages of +/- 2% are generally acceptable. For this project, f_{OSC} is equal to 8 MHz and asynchronous normal transmit mode is used. For this combination a baud rate of 38400 bps will have an error percentage of 0.2%. For

asynchronous normal transmit mode, the clock signal generated from the baud rate generator will have frequency equal to $f_{OSC}/[16 \bullet (UBRRn + 1)]$.

- *Analog to Digital Converter:* The ADC in the AVR microcontroller is used to convert analog voltage to a digital value. There are several analog signals that one might like to measure. For example, this tag carries an on board sensor to measure temperature. The built in ADC in ATmega328P has 10 bit resolution. The ADC is attached to an 8-channel analog multiplexer which allows up to 8 single ended voltage inputs to be connected to the AVR. Single ended voltage input means that the voltage input is with respect to ground (0V). The ADC has a separate supply voltage pin AV_{CC}. This voltage must not differ by more than $\pm 0.3V$ from V_{CC}. In most cases V_{CC} is wired to AV_{CC}. There are several signals that can be selected as the reference voltage for the ADC. The reference voltage (V_{REF}) determines the range of the ADC. The available options for V_{REF} are AV_{CC}, the internal 1.1V reference or the external AREF pin.
  For single ended conversion, the result from ADC is-

$$ADC = \frac{V_{IN} \bullet 1024}{V_{REF}} \qquad (3.1)$$

  where V_{IN} is the voltage on the channel selected (out of the 8 channels available) for conversion. The ADC clock is derived from the system clock. The ADC clock should have a frequency between 50 kHz and 200 kHz to get maximum resolution. This can be done by using the prescaler contained within the ADC module to scale the system clock to a frequency within the abovementioned frequency range.

3.1.2 CC2520 2.4 GHz IEEE 802.15.4/ZigBee compliant RF transceiver

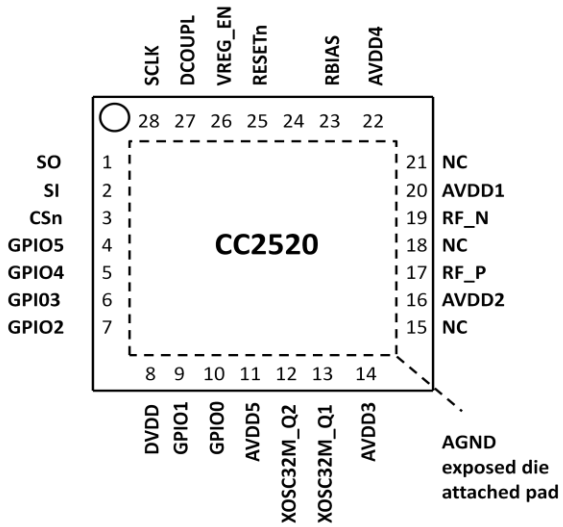


Figure 3.4: Pinout of CC2520 [15]

CC2520 is a direct conversion transceiver operating in the 2.4 GHz ISM band. It complies with the IEEE 802.15.4 PHY specification. The transceiver has a data rate of 250 kbps and a chip rate of 2 Mchips/s. It contains a SPI interface through which it can be connected to a microcontroller. The microcontroller is configured to be the master while CC2520 is configured to be the slave. Through the SPI interface instructions can be sent to CC2520. The instructions are byte oriented and the number of bytes required per instruction is greater than or equal to one. The SPI data mode '0' should be used when communicating with CC2520. The pin configuration of CC2520 is shown in fig. 3.4 and the connection between it and the microcontroller in the ZT-500 tag is shown in fig. 3.5.

Key features of CC2520 are as follows [15]-

- IEEE 802.15.4 compliant DSSS baseband modem
- Long communication range-400m when line of sight present
- Low Power: Rx current (receiving frame, -50 dBm) 18.5 mA, Tx current 33.6 mA @5dBm, power down current less than 1 μA
- Output power is programmable
- Automatic clear channel assessment for CSMA/CA
- Automatic CRC insertion capability present
- 4 wire SPI
- 6 configurable general purpose input/output pins
- RSSI/LQI indication
- Operating temperature range: -40 ºC to 125 ºC
- Operating voltage range: 1.8V to 3.8V

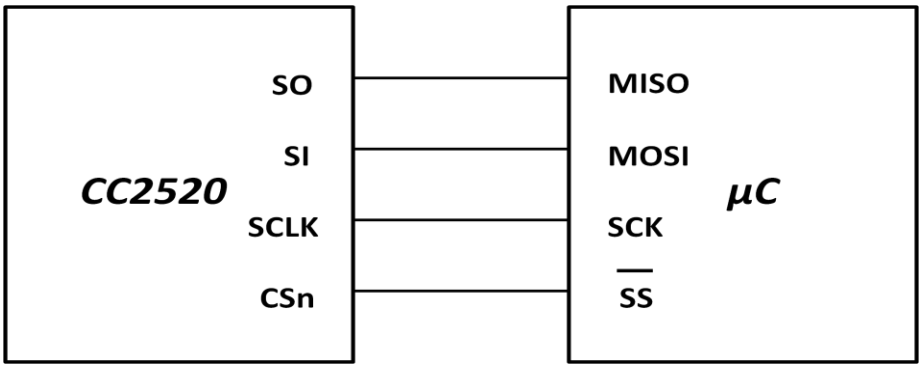


Figure 3.5: Interconnection between CC2520 and the ATmega328P microcontroller

Through the programmable configuration registers of CC2520, the following key parameters can be programmed [15]-

- Receive/Transmit mode
- RF channel selection
- RF output power
- Active mode/Low Power Mode 1 (LPM1)/ Low Power Mode 2 (LPM2)
- Crystal oscillator on/off

- Clear Channel Assessment mode
- Encryption/authentication algorithm

3.1.3 CC2591 2.4 GHz RF front-end

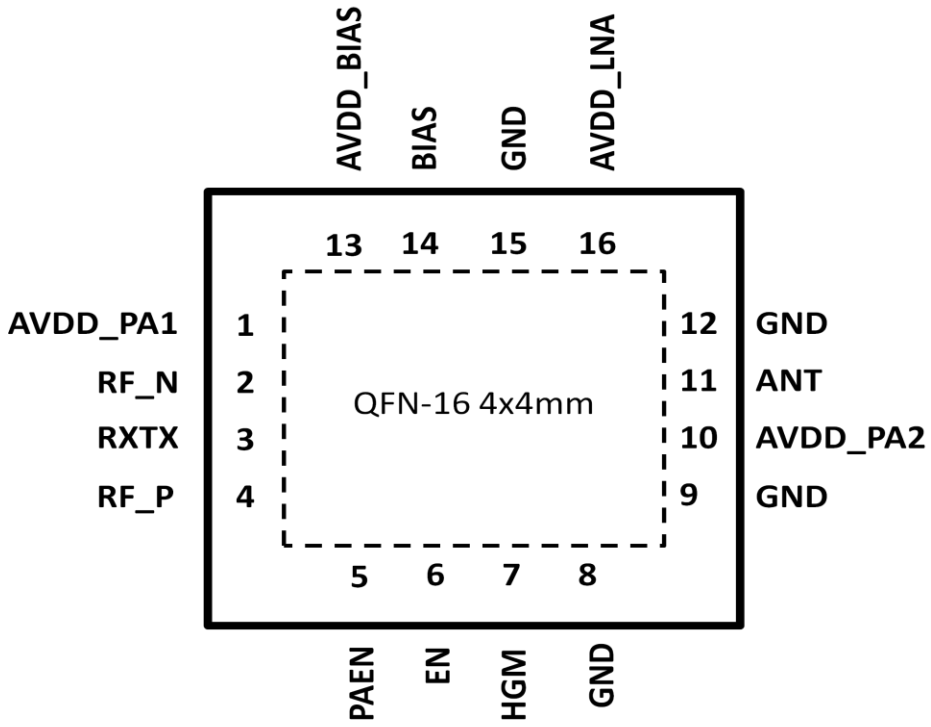


Figure 3.6: Pinout of CC2591 [21]

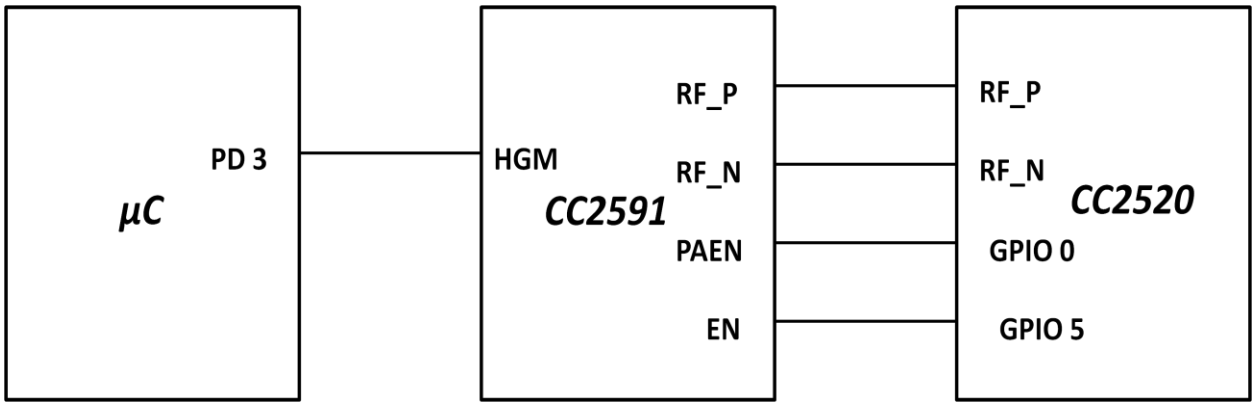


Figure 3.7: CC2591 connections on the ZT-500 Tag

CC2591 is used for extending the range of 2.4 GHz low power RF transceivers. It houses a power amplifier (PA) and a low noise amplifier (LNA) and provides a 2.4 GHz RF front-end. Fig. 3.6 shows the pin configuration of CC2591.

Some of the key features of CC2591 are [21]-

- Output power up to 22 dBm
- Typical improved sensitivity of 6 dB
- Low transmit current consumption: 100 mA for 20 dBm output at V_{cc}=3V
- Low receive current consumption: 3.4 mA for high gain mode, 1.7 mA for low gain mode
- LNA gain can be controlled digitally by using HGM pin
- Operating temperature range: -40 ºC to 85 ºC
- Operating voltage range: 2V to 3.6V

The CC2591 chip is controlled both by CC2520 and ATmega328P on the ZT-500 tag as shown in fig. 3.7. The controlling logic for CC2591 is shown in table 3.4.

PAEN	EN	HGM	Mode of Operation
0	0	X	Power down
0	1	0	RX LGM
0	1	1	RX HGM
1	0	X	TX
1	1	X	Not allowed

Table 3.4: Controlling logic for CC2591 [21]

3.1.4 32.768 kHz crystal oscillator

The 32.768 kHz oscillator is used to implement a real time clock on the ZT-500 tag. This oscillator is preferred for realizing a real time clock because if its frequency is divided by 2 fifteen times then the frequency of the resulting clock signal will be 1 Hz. This oscillator provides the clock source for Timer/Counter2 of ATmega328P in the asynchronous mode. The overflow interrupt in Timer/Counter2 is generated twice every second on the ZT-500 tag. To achieve this, following operations are carried out-

1. The frequency of the clock signal from the 32.768 kHz oscillator is divided by 64 by using the prescaler inside Timer/counter2. The frequency of the resulting clock signal is equal to 512 Hz.
2. The resulting clock source from step 1 is used to increment the counter by one after every clock cycle.
3. The overflow interrupt for Timer/Counter2 is enabled.

As Timer/Counter2 is an 8-bit module, an overflow will occur after every 2^8 or 256 clock cycles. Therefore an overflow interrupt is generated every half a second because the frequency of the clock source is 512 Hz. This way a real time clock can be implemented on the ZT-500 tag.

3.1.5 Temperature Sensor

The ZT-500 tag also carries an analog sensor on board for measuring temperature. This sensor measures the temperature and on the basis of this measurement generates a single ended voltage signal. The sensor is connected to the seventh channel of the ADC in the ZT-500 tag.

3.1.6 Energy Module

To provide energy to the ZT-500 tag, two AA alkaline batteries are used. The voltage provided by each battery is 1.5V and therefore when the two batteries are connected in series a 3V voltage source can be made available to the tag. All of the active components on the tag work well when a voltage equal to 3V is applied across them. These AA alkaline batteries are cheap, easily available in the market and can have a capacity up to 2900 mAh each [22].

3.2 RS-232 Adaptor

The USART module in the AVR is often used to communicate with a laptop or a PC through the RS-232 port. The AVR's voltage levels are between 0 and V_{CC} while the RS-232 communication uses voltage between -3V to -15V for logic one and voltage between +3V to +15V for logic zero. Thus there is a need of a level converter between the AVR and the computer. In this project *Tagsense's* RS-232 adaptor is used for this level conversion.

3.3 USB to Serial Convertor

As most of the laptops today do not contain a RS-232 port, a convertor is needed to convert the RS-232 signal to a USB signal so that the USB port in the laptop can be used for RS-232 communication. For this project Gigaware's USB to serial convertor has been used.

3.4 AVR mkII Programmer

The AVR mkII programmer is used to program the ATmega328P microcontroller through the In-System Programming (ISP) interface. Some devices like the microcontrollers have the ability to be programmed while they are installed in a system. This ISP capability allows the microcontrollers to be reprogrammed as and when needed by an end user of the system in which the microcontroller is integrated. The ISP interface used by the AVR mkII programmer to program the ATmega328P microcontroller is the SPI interface. The AVR mkII programmer does not provide power to the microcontroller and therefore the microcontroller needs its own power source before it can be programmed. The programmer takes the hexadecimal code from the laptop and transfers it to the application flash section of the flash memory inside the microcontroller through the SPI interface.

3.5 Bosch Sensortec's BMA150 Accelerometer

The BMA150 is a triaxial acceleration sensor from *Bosch Sensortec*. The output from the sensor is digital. It can also measure the absolute temperature. The pin configuration of BMA150 is shown in fig. 3.8.

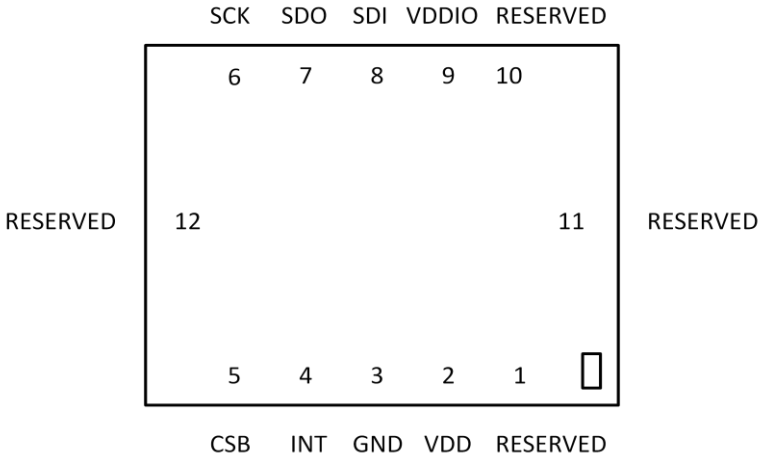


Figure 3.8: Pinout of BMA150[23]

The key features of BMA150 are[23]-

- It measures acceleration in three perpendicular axes
- The data transfer from accelerometer can take place through SPI (4-wire, 3-wire) or I^2C
- Low current consumption: typically 200µA in normal mode and 1µA in standby mode
- Operating temperature range: -40ºC to 85ºC
- Operating voltage range: 2.4V to 3.6V

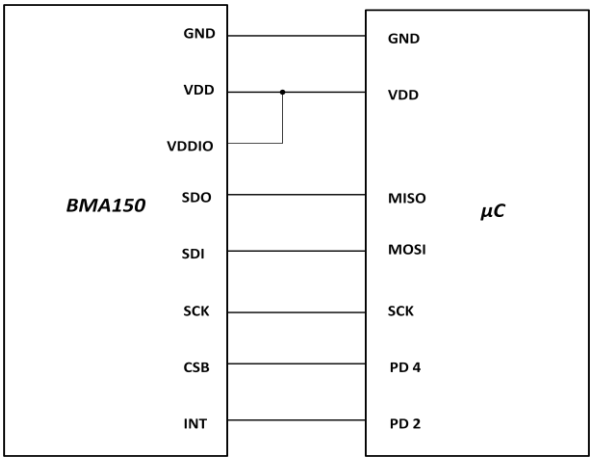


Figure 3.9: Interconnection between BMA150 and ATmega328P

The SPI 4-wire interface is used in this project to configure BMA150. The microcontroller controlling BMA150 should be configured as the master and BMA150 should be configured to be the slave. The SPI data mode used should be mode '3' when communicating with BMA150. The interconnection between BMA150 and ATmega328P is shown in fig. 3.9.

By using the programmable registers inside BMA150, the following important features can be programmed-

- Acceleration range
- Bandwidth of the digital filter filtering the ADC output data
- Sleep mode duration
- Interrupt settings

BMA150 switches from sleep mode to normal mode after the specified sleep mode duration and acquires acceleration data. It then checks whether any of the active interrupt conditions are fulfilled or not. If none of the interrupt conditions are met then BMA150 goes back to sleep mode. BMA150 gives the option of choosing among various interrupt conditions. Multiple interrupt conditions can be specified simultaneously.

4 Software Development

The only software used in this project to write the code for microcontroller is AVR Studio 4. There are several libraries used to develop the code. The AVR Studio 4 is a freeware and can be downloaded from www.atmel.com website. The algorithms used in the realization of AVR components like ADC, SPI, etc. and the interfacing of accelerometer module with the tag are discussed in this section. The ZigBee frame format, the *TagSense* protocol and the implementation of transmit and receive functions for CC2520 are also discussed in this section.

4.1 AVR Studio 4

AVR Studio is an Integrated Development Environment (IDE) for writing and developing Atmel 8-bit AVR applications in Windows NT, Windows 2000, Windows XP, Windows Vista and Windows 7 environments [24].

AVR Studio provides an editor which allows the programmer to write code in C or assembly language. It also provides an option to choose from multiple debugging platforms and supports several AVR microcontrollers. For this project the programming language used is C, the debugging platform used is AVR Simulator 2 and the microcontroller used is ATmega328P.

AVR Studio uses WinAVR for compiling and linking the main program and various libraries. WinAVR is a collection of open source executable software development tools for the Atmel AVR microcontroller running on Windows. It includes the GNU GCC compiler for C and C++. It contains the following software development tools [25]-

- Compilers
- Assembler
- Linker
- Librarian
- File Converter
- Other file utilities
- C Library
- Programmer software
- Debugger
- In-Circuit Emulator software
- Editor/IDE
- Many support utilities

After the program is compiled and debugged using AVR Studio, the binary image can be downloaded on to a microcontroller using an AVR programmer. The AVR programmer used for this project is AVR mkII. The AVR programmer can be connected to the USB port of a PC

running AVR Studio on one end and to the microcontroller through the ISP interface on the other end. The ISP frequency used for programming the AVR should be less than ¼ of the clock frequency used for the microcontroller. The clock used for ATmega328P in this project has a frequency of 8 MHz.

4.2 Libraries

The main libraries used in this project are-

- AVR Input/output library
- AVR Interrupt library
- AVR sleep library
- AVR EEPROM library
- AVR program memory library
- AVR boot loader support library

4.2.1 AVR Input/Output Library

The *<avr/io.h>* library includes the appropriate input-output definitions for the AVR microcontroller specified by the *–mmcu=* compiler command-line switch. This command is generated by AVR Studio when the program is compiled. The *<avr/io.h>* file uses the *<avr/ioXXX.h>* file, where *XXX* denotes the AVR device in use, to access details regarding the microcontroller. Some register names are common to all AVR devices and are defined within *<avr/common.h>*, which is included in *<avr/io.h>* [26].

4.2.2 AVR Interrupt Library

The *<avr/interrupt.h>* library provides functions which help in handling interrupts. In the AVR-GCC environment the interrupt routines with predefined names are located at specific memory locations. The AVR maintains a vector table which links the interrupt name to the memory location. A particular routine can be called upon by using the appropriate interrupt name [27].

There is a global interrupt flag which is maintained in the *I bit* of the *Status register (SREG)*. This bit should be set to logic one if interrupts are to be used. Otherwise it should be set to zero. Global interrupt flag can be set by using the *sei( )* macro and can be cleared by using the *cli( )* macro. The *ISR( )* macro is used to define the routine that should be called when a particular interrupt occurs. For example, the handler for the ADC vector can be defined in this way [27]-

```
#include <avr/interrupt.h>

ISR(ADC_vect)
{
        //user code here
}
```

Sometimes when the interrupts are enabled (global interrupt flag is set to '1'), an unexpected interrupt may occur for which no interrupt handler has been defined. By default when such a situation occurs, the device is reset. This action can be overridden by using the *BADISR_vect* interrupt vector in the following way [27]-

```
#include <avr/interrupt.h>

ISR(BADISR_vect)
{
        //user code here
}
```

4.2.3 AVR Sleep Library

The AVR should be put into a sleep mode as often as possible to save power. The *<avr/sleep.h>* header file contains functions which can be used to put the AVR device into sleep mode. There are six different sleep modes available in ATmega328P. The sleep mode to be used can be set by using the *set_sleep_mode( )* macro. The *sleep_enable( )* macro can be used to set the *Sleep Enable (SE)* bit in the *Sleep Mode Control Register (SMCR)*. This bit should be set just before the sleep instruction is executed and should be cleared immediately after the AVR comes out of sleep mode. The *sleep_disable( )* macro can be used to clear the *SE bit* [28].

The device can be put into sleep mode by using the *sleep_cpu( )* macro and the *sleep_bod_disable( )* macro can be used to turn off the *Brown-out Detector (BOD)* which actively monitors the power supply voltage during a sleep period [28]. Before the device is put to sleep, the global interrupt flag should be set using the *sei( )* macro. This is done because the AVR can exit the sleep mode only when an interrupt occurs. The interrupts that can be used as wake-up sources for different sleep modes are shown in table 4.1. The sleep mode used for this project is the *Power-save mode* and the wake-up source used is the *Timer2 interrupt*.

	Wake-up Sources						
Sleep Mode	**INT1, INT0 and Pin Change**	**TWI Address Match**	**Timer2 (in asynchronous mode)**	**SPM /EEPROM Ready**	**ADC**	**WDT**	**Other I/O**
Idle	✓	✓	✓	✓	✓	✓	✓
ADC Noise Reduction	✓	✓	✓	✓	✓	✓	
Power-down	✓	✓				✓	
Power-save	✓	✓	✓			✓	
Standby	✓	✓				✓	
Extended Standby	✓	✓	✓			✓	

Table 4.1: Wake-up Sources in Different Sleep Modes [20]

When *Timer2* is used as the wake-up source in the *Power-save mode* and the AVR has to go back to the sleep mode then certain precautions should be taken otherwise there will be multiple interrupts within one clock cycle. To avoid this from happening, the following steps can be followed [20]-

1. Write a value to TCCR2x, TCNT2 or OCR2x.
2. Wait until the corresponding Update Busy Flag in ASSR returns to zero.
3. Enter the Power-save mode.

The following example shows how to enter the *Power-save mode* when using the *Timer2 interrupt* as the wake-up source-

```
#include <avr/interrupt.h>
#include <avr/sleep.h>

.....

sei();  //set global interrupt flag
set_sleep_mode(SLEEP_MODE_PWR_SAVE);  //set sleep mode to power-save
while(1)
{
        ADCSRA&=(~(1<<ADEN));  //Disable the ADC
        PRR|=(1<<PRADC);  //Shutdown the ADC
        sleep_enable();  //set the SE bit
        TCCR2B = TCCR2B;  // dummy write
        while( ASSR & 1<<TCR2BUB );  //wait for the update busy flag to return to zero
        sleep_bod_disable();  //disable the BOD
        sleep_cpu();  //enter sleep mode
        sleep_disable();  //clear the SE bit
        //rest of the user code here
}
```

4.2.4 AVR EEPROM Library

Most of the AVR microcontrollers contain a certain amount of internal EEPROM memory. This memory is non-volatile which means that it can retain its data even when no power is available to the microcontroller. It has a limited lifespan- in ATmega328P it will last for at least 100,000 write cycles, there is no restriction on the number of reads.

The EEPROM memory is accessed through special registers inside AVR. These registers tell the microcontroller the address to be written to and the data to be written. There are certain flags that tell the AVR whether the data is to be written to or read from the EEPROM [29].

The <*avr/eeprom.h*> library provides functions for writing to and reading from the EEPROM. The following routines are available in the <*avr/eeprom.h*> header file[29]-

- uint8_t eeprom_read_byte (const uint8_t *addr) //read a single byte from EEPROM
- void eeprom_write_byte (uint8_t *addr, uint8_t value) //write a single byte to EEPROM
- uint16_t eeprom_read_word (const uint16_t *addr) //read a word (two bytes) from EEPROM
- void eeprom_write_word (uint16_t *addr, uint16_t value) //write a word to EEPROM
- void eeprom_read_block (void *pointer_ram, const void *pointer_eeprom, size_t n) //read a block (multiple bytes) of data from EEPROM
- void eeprom_write_block (void *pointer_eeprom, const void *pointer_ram, size_t n) //write a block of data to EEPROM

One should make sure that the *EESAVE fuse* is programmed so that the data is saved in the EEPROM when the power to the AVR is switched off. The fuses can be set using the fuses tab in AVR Studio. The *EESAVE fuse* should be checked and the *PROGRAM button* at the bottom of the tab should be clicked to program the fuse as shown in fig. 4.1.

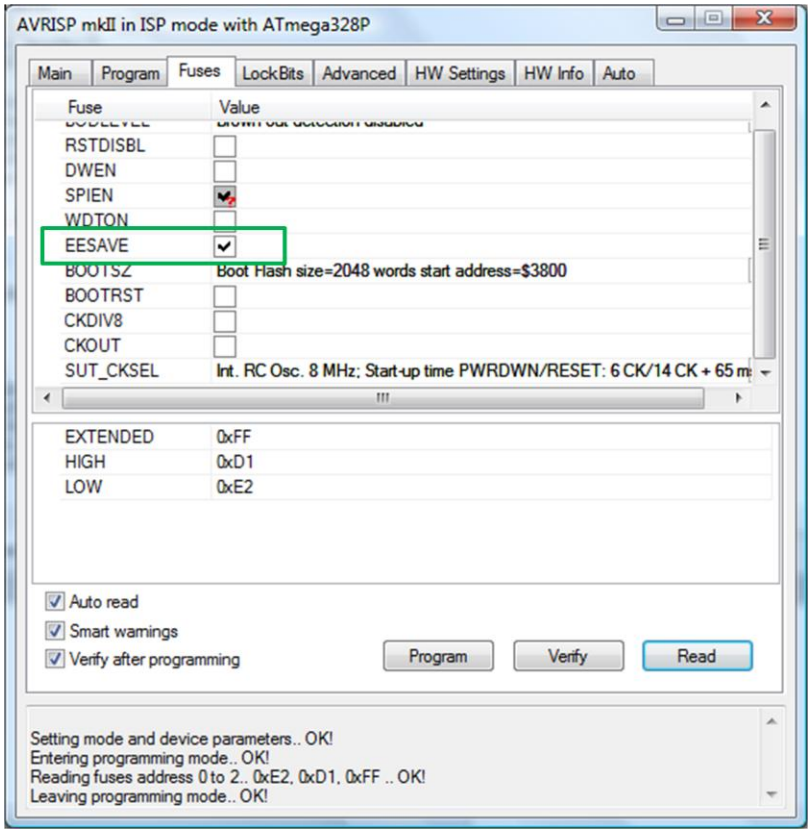


Figure 4.1: Programming the EESAVE Fuse

4.2.5 AVR Program Memory Library

The *<avr/pgmspace.h>* library provides various macros which can be used to read data from the program memory. When there is shortage of SRAM, the program memory can be used to store data as it is quite large compared to SRAM and EEPROM [30]. The program memory in ATmega328P is 32K bytes. In this project 8K bytes of the program memory are used for data logging purposes. The only function used from the *<avr/pgmspace.h>* header file in this project is the *pgm_read_byte_near( )* function which can be used to read a byte of data from the program memory by specifying a 16 bit (near) address.

4.2.6 AVR Boot Loader Support Library

The boot loader program resides in the boot loader section of the flash (program) memory and can write to the entire flash, including the boot loader section[20]. It can grab data from any interface (USART, SPI, etc.) and write that data to the flash memory and thus has the capability to change the program loaded into the flash memory. In this project we use boot loader only to store data in the flash memory. The size of the boot loader can be configured with the help of fuses. The *<avr/boot.h>* library provides macros that can be used to exploit the boot loader support functionality available to certain AVR microcontrollers.

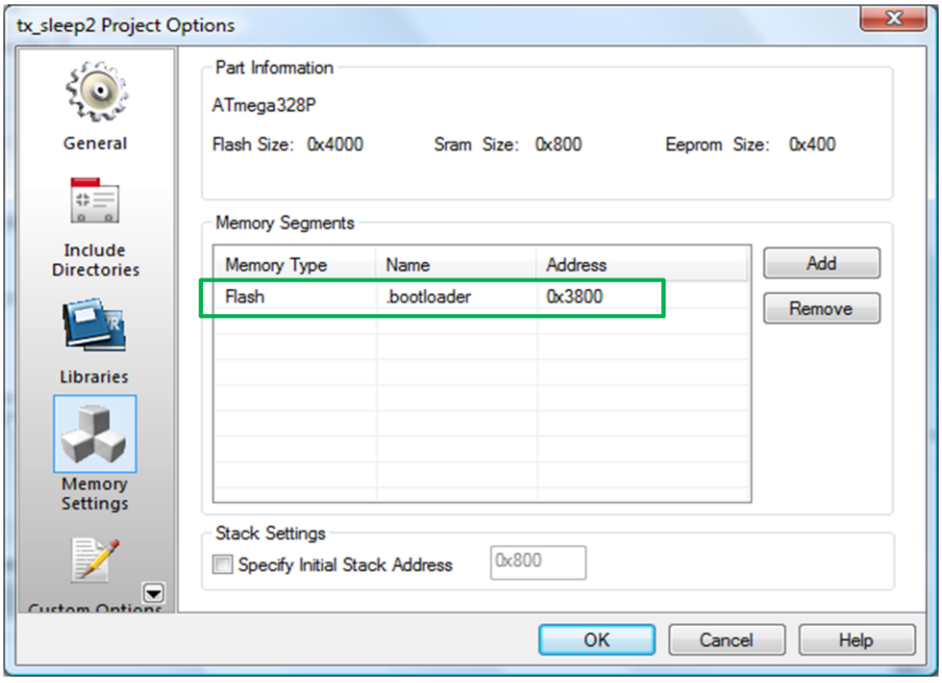


Figure 4.2: Setting the starting address for the .bootloader section

The flash is divided into two fixed sections- Read-While-Write (RWW) and No Read-While-Write (NRWW). The main difference between the two is that when erasing or writing a page in the RWW section, the NRWW section can be read while when a page is being written to or

erased from the NRWW section the CPU is halted and the RWW section cannot be read. The boot loader section is always in the NRWW section[20].

To store a function or a variable in the boot loader section, it should be defined with the *BOOTLOADER_SECTION* attribute as shown below [30] -

*void boot_program_page (uint32_t page, uint8_t *buf) BOOTLOADER_SECTION;*

The BOOTLOADER_SECTION attribute is defined in *<avr/boot.h>*. The above function declaration tells the compiler to store the function *boot_program_page( )* in the *.bootloader section* of the flash memory. The memory address where this section is located can be configured in AVR Studio as shown in fig. 4.2. Please note that the address shown in fig. 3.2 is in terms of words rather than bytes. Therefore the address 0x3800 means that the *.bootlader section* starts from byte number 0x7000 in the flash memory.

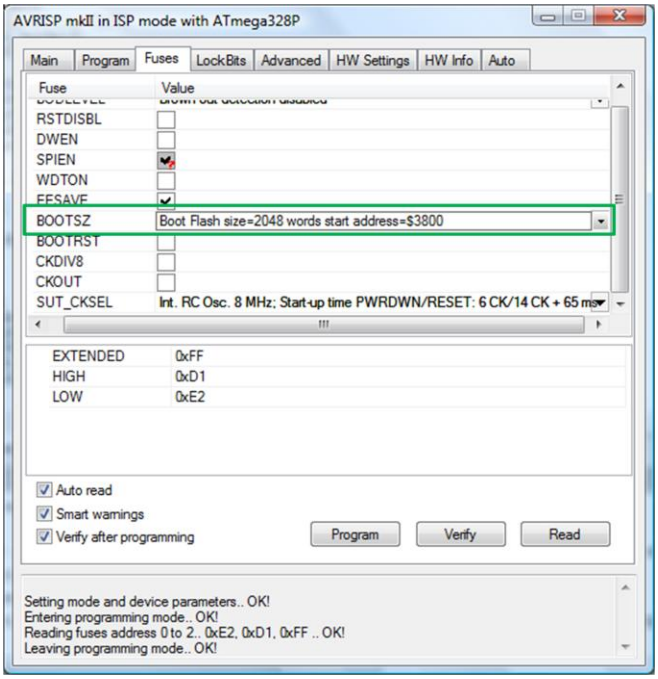


Figure 4.3: Setting the size of the .bootloader section

The size of the *.bootloader section* can be set using the *BOOTSZ fuses* as shown in fig. 3.3. For ATmega328P the RWW section extends from 0x0000-0x37FF and the NRWW section extends from 0x3800-0x3FFF (the memory is addressed in words) [20]. In this project the whole NRWW section is used as the *.bootloader section* as can be seen from fig. 4.3.

The flash memory is written in a page by page fashion. Before programming a page in the flash memory with the data stored in a temporary page buffer, the contents of that page must be erased. The temporary page buffer is filled one word at a time using the *Store Program Memory*

(SPM) instruction. There are two different ways in which a page can be written to the flash memory [20] -

Alternative 1:

- Fill temporary page buffer
- Perform a page erase
- Perform a page write

Alternative 2:

- Perform a page erase
- Fill temporary page buffer
- Perform a page write

In this project alternative 2 is used to write a page to the flash memory.

4.3 Implementation of AVR components

The AVR components implemented in this project are-

1. ADC
2. Timer/Counter2
3. SPI
4. USART

4.3.1 ADC

The ADC in this project is implemented in the following way-

1. The PRADC bit in PRR is set to '0'.
2. The ADCEN bit in ADCSRA is set to '1' to enable the ADC.
3. The division factor between the system clock frequency and the input clock to the ADC is specified by setting the ADPS [2:0] bits. The input clock to the ADC should have a frequency between 50 kHz and 200 kHz to get maximum resolution.
4. The ADMUX register is used to set the reference voltage and the ADC channel to be used for conversion.
5. The ADSC bit in the ADCSRA register is set to '1' to start the conversion.
6. Wait until the conversion is complete. When the conversion is complete the ADSC bit in ADCSRA is set to '0' by the hardware.
7. The result of the conversion is stored in ADCL and ADCH. The ADCL should be read first and then the ADCH.

The ADC in addition to converting the analog voltage signals from sensors to digital signals can also be used to calculate the V_{CC} applied to the AVR. For single ended conversion the result from the ADC is given by equation (3.1) which is shown below-

$$ADC = \frac{V_{IN} \bullet 1024}{V_{REF}} \qquad (3.1)$$

The ADC channel used in the conversion should be the internal 1.1 V reference voltage signal and the reference voltage used should be AV_{CC} which is equal to V_{CC} in this project. Then the result from the ADC is given by-

$$ADC = \frac{(1.1) \bullet 1024}{V_{CC}} \qquad (4.1)$$

From equation (4.1) it can be seen that the V_{CC} can be calculated easily from the result of the ADC conversion.

4.3.2 Timer/Counter2

To implement the Timer/Counter2 in this project, the following steps were involved-

1. Disable all the Timer/Counter2 interrupts by setting TIMSK2 register to 0x00.
2. Enable the asynchronous mode by writing '1' to AS2 bit in ASSR register.
3. Set the initial counter value to zero by setting TCNT2 register equal to 0x00.
4. Specify the prescaler division factor by setting the CS2 [2:0] bits in TCCR2B register.
5. Wait for the registers to update. When the Timer/Counter2 is running in asynchronous mode and the TCNT2 register is updated then the TCNT2UB bit in ASSR register becomes set. This bit is cleared by the hardware when the TCNT2 register has been updated. The TCR2BUB bit in ASSR register works in a similar way when TCCR2B register is updated.
6. Clear the TOV2 bit in the TIFR2 register by writing '1' to this bit. This bit will be set by the hardware when an overflow occurs in Timer/Counter2.
7. Enable the Timer/Counter2 overflow interrupt by writing '1' to the TOIE2 bit in the TIMSK2 register.

4.3.3 SPI

The AVR is used as the master when using the SPI interface in this project. The SPI is implemented in the following way-

1. Set MOSI, SCK and CS pin of the AVR as outputs and MISO as input.
2. Enable the SPI by writing '1' to SPE bit in SPCR register and configure the SPI to be master by writing '1' to MSTR bit also in SPCR register.

3. Specify the prescaler division factor by using SPR1, SPR0 bits in the SPCR register and the SPI2X bit in the SPSR register.
4. Specify the SPI mode by using the CPOL and CPHA bits in the SPCR register.
5. Write the data to be transmitted to the SPDR register.
6. Wait for the data transmission to complete. The SPIF flag in the SPSR register will be set when the data transmission is over.
7. The received data can be read from the SPDR register.

4.3.4 USART

The USART is used in the asynchronous mode in this project. The functionality of USART is implemented in the following way-

1. Specify the baud rate to be used in the UBRR0H and UBRR0L registers. The baud rate should be chosen in such a way that the percentage error is not more than +/- 2%.
2. Set the USART to be used in the asynchronous mode by using the UMSEL01 and UMSEL00 bits in the UCSR0C register.
3. Set the number of parity bits, stop bits and data bits by using the UCSR0C register.
4. Enable the transmitter and the receiver by writing '1' to the TXEN0 and RXEN0 bits in the UCSR0B register.
5. To generate an interrupt when USART reception is complete, write '1' to the RXCIE0 bit in the UCSR0B register and the RXC0 bit in the UCSR0A register.

4.4 The ZigBee Frame Format

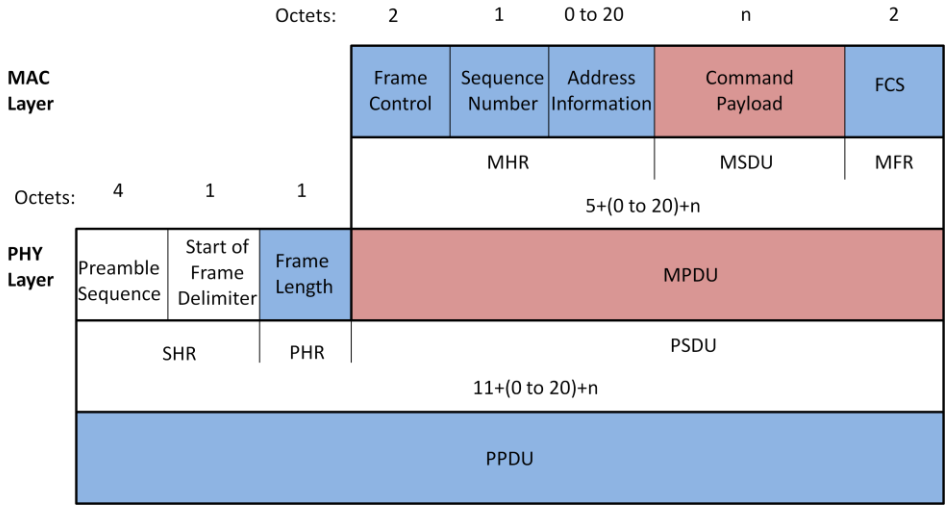


Figure 4.4: Schematic view of the IEEE 802.15.4 Frame Format [31]

The ZigBee protocol is used by the CC2520 transceiver to transmit and receive data over the air. The IEEE 802.15.4 frame format is shown in fig. 4.4. The PHY layer consists of the *Synchronization Header (SHR)*, the *PHY Header (PHR)* and the *PHY Service Data Unit (PSDU)*.

The CC2520 transceiver takes care of generation and automatic transmission of *SHR*, transmission of the number of bytes specified by the frame length field and calculation and transmission of *Frame Check Sequence (FCS)* (which can be disabled). The user has to take care of the *MAC Protocol Data Unit (MPDU)* except for the *FCS* [15].

Bits:0-2	3	4	5	6	7-9	10-11	12-13	14-15
Frame Type	Security Enabled	Frame Pending	Acknowledgement Request	Intra PAN	Reserved	Destination Addressing Mode	Reserved	Source Addressing Mode

Figure 4.5: Format of the Frame Control Field [31]

The format of the *Frame Control Field (FCF)* is shown in fig. 4.5. The value of the *FCF* transmitted in this project is equal to '0x0820'. As the MSB is transmitted first, this means that the first sub-field transmitted is the *'Source Addressing Mode'*. The value and meaning of the subfields transmitted inside FCF are (MSB is the rightmost bit and LSB is the leftmost bit)-

- Source Addressing Mode=0b00; Source PAN Identifier and addressing field not present.
- Destination Addressing Mode=0b10; Destination address field contains a 16 bit short address.
- Intra PAN=0b0; Frame has to be sent to another PAN.
- Acknowledgement Request=0b1; Acknowledgement is requested from the recipient device.
- Frame Pending=0b0; There is no more data to be sent to the recipient device.
- Security Enabled=0b0; Frame is not cryptographically protected by the MAC sub-layer.
- Frame Type=0b000; Frame is a beacon frame.

The field next to *FCF* in the *MPDU* specifies the sequence number of the frame and is 1 byte in length. The format of the *Addressing Information Field* is shown in fig. 4.6.

Octets: 0/2	0/2/8	0/2	0/2/8
Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address

Figure 4.6: Format of the Addressing Information Field [31]

As the destination addressing mode sub-field in the *FCF* is non zero and the source addressing mode sub-field in the *FCF* is zero, only the destination PAN identifier and destination address sub-fields are present in the address information field. In this project, the destination PAN identifier sub-field is equal to the broadcast PAN identifier '0xFFFF' and the destination address sub-field is equal to '0xFFFF'-the broadcast address, which means that any device listening to the channel should accept this frame.

4.5 CC2520 Programming

The CC2520 transceiver is controlled by the microcontroller using the SPI interface. When CC2520 and CC2591 are used together, the registers shown in table 4.2 should be updated from their default value every time the transceiver is brought back to active mode from LPM2.

Register Name	Address (hex)	New value (hex)
TXPOWER	030	See table 3.3
CCACTRL0	036	F8
MDMCTRL0	046	85
MDMCTRL1	047	14
RXCTRL	04A	3F
FSCTRL	04C	5A
FSCAL1	04F	2B
AGCCTRL1	053	16
ADCTEST0	056	10
ADCTEST1	057	0E
ADCTEST2	058	03
GPIOCTRL0	024	46
GPIOCTRL5	025	47
GPIOPOLARITY	026	1E
TXCTRL	031	C1

Table 4.2: Registers that need update from their default value when using CC2520-CC2591 combination [16]

The *GPIO0* pin and the *GPIO5* pin in CC2520 are connected to the *PA_EN* pin and the *EN* pin in CC2591 respectively and therefore the value of registers *GPIOCTRL0* and *GPIOCTRL5* is updated. Also, the polarity of *GPIO0* and *GPIO5* should be negative and therefore the value of *GPIOPOLARITY* is updated to 0x1E. The transmit power to be set can be chosen from table 4.3. The values shown in table 3.3 were measured on the CC2520-CC2591 EM reference design with a 50Ω load at +3.0V, +25ºC and f_C=2440MHz.

TXPOWER	Power [dBm]	Current [mA]
0xF9	17	136
0xF0	16	121
0xA0	14	102
0x2C	11	78
0x03	-1	57
0x01	-8	55

Table 4.3: Power table when using CC2520-CC2591 combination[16]

The frequency channel to be used can be set using *FREQCTRL* register in CC2520. CC2520 supports the frequency range from 2394 MHz to 2507 MHz. The value specified in the *FREQCTRL* register is offset to 2394 MHz to calculate the actual RF frequency to be used. The frequency range specified in IEEE 802.15.4-2006 is 2405 MHz to 2480 MHz with 16 channels 5

MHz apart. The channels are numbered 11 through 26. The only valid values of *FREQCTRL* register when using an IEEE 802.15.4-20006 compliant system are [11+5(channel number -11)] [15].

4.5.1 CC2520 Receiver

The CC2520 receiver can be turned on and off by using the *SRXON* and *SRFOFF* command strobes. The received frame is stored in the *RX FIFO* buffer. To find out whether the frame reception is complete or not, the *RX_FRM_DONE* exception bit can be read from the *EXCFLAG1* register. The exception will be raised if the frame reception is complete. A '0' should be written manually to the *RX_FRM_DONE* exception bit to clear it [15].

The receiver is turned on when CC2520 is sensing the channel before transmission. The *RSSISTAT* register indicates whether the RSSI value is valid or not. The RSSI value is valid only after eight symbol periods from the time when the receiver was turned on. The RSSI value can be read from the RSSI register. To determine whether the channel is being used or not, the RSSI value is compared with the threshold set in the *CCACTRL0* register. If RSSI is less than the threshold then the channel is determined to be free otherwise busy (the decision also depends on the value of *Clear Channel Assessment (CCA) hysteresis* specified in the *CCACTRL1* register) [15].

The number of bytes in the *RX FIFO* buffer is stored in the *RXFIFOCNT* register. The *RXBUF* command strobe can be used to read the data in *RX FIFO*. The CC2520 transceiver also runs a frame filtering algorithm to reject frames not intended for the device [15]. In this project the frame filtering algorithm has been turned off.

4.5.2 CC2520 Transmitter

CC2520 uses *TX FIFO* buffer to store the frame to be transmitted. *TX FIFO* can store a maximum of 128 bytes and only one frame at a time. The data can be stored in the *TXFIFO* by using the *TXBUF* or *TXBUFCP* command strobe. The transmission of a frame can be started by using the *STXON* or *STXONCCA* command strobes. The CC2520 transmitter can be turned off by using the *SRFOFF* command strobe. CC2520 does not delete the contents of the *TXFIFO* after the frame transmission is complete. Therefore a frame can be retransmitted by simply issuing a *STXON* or *STXONCCA* command strobe again. When the transmission is complete, the *TX_FRM_DONE* exception is raised in the EXCFLAG0 register. A '0' should be manually written to the *TX_FRM_DONE* exception bit to clear it. The *TX FIFO* buffer can be emptied by using the *SFLUSHTX* command strobe [15].

The CC2520 implements the *CSMA-CA* protocol specified in IEEE 802.15.4. It uses the *CCA* status signal to indicate whether the channel is clear for transmission or not. The *CCA* signal can be updated in two ways-at every new RSSI sample or on *SSAMPLECCA* and *STXON* command strobes. The latter one is used when using *slotted CSMA-CA* while the former one is used when

using *unslotted CSMA-CA*. The *SAMPLED_CCA* bit in the *FSMSTAT1* register can be read to find out whether the transmission can begin or not. The transmission can also take place without the use of *CSMA-CA* [15]. In this project *slotted CSMA-CA* is used for transmission. The TX flow diagram in CC2520 is shown in fig. 4.7.

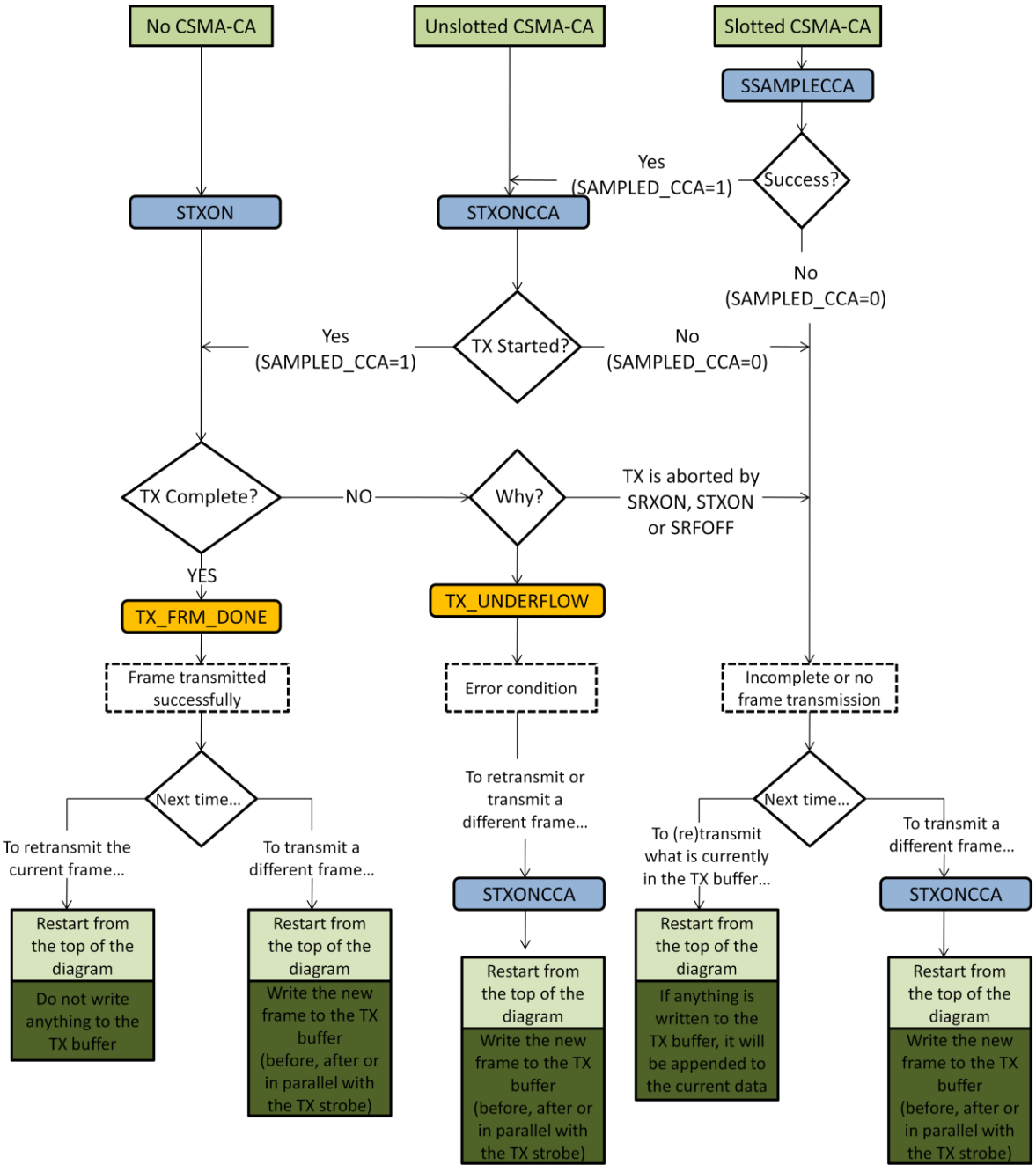


Figure 4.7: TX Flow Diagram in CC2520 [15]

4.6 Interfacing the Accelerometer

The BMA150 accelerometer is interfaced with the ATmega328P microcontroller using 4-wire SPI interface in the following way-

1. The AVR is configured to be the master while BMA150 is configured to be the slave.
2. SPI Mode 3 is used for communication between the AVR and BMA150.
3. The value of operational registers at address 0x14 and 0x15 should be modified if required to set the bandwidth of the digital filter filtering the ADC output data and the sleep phase duration after each automatic wake-up respectively.
4. Read the value of operational registers at addresses 0x02-0x07 to obtain the acceleration data in the three perpendicular axes.

5 Graphical User Interface

The Graphical User Interface (GUI) in this project was made using Apache web server and Perl scripting language. Dynamic web pages were developed using Common Gateway Interface (CGI) scripts which are written in Perl in this project. CGI scripts can also be written in many other languages. Static web pages are simply HTML text files which do not change until the file itself is changed. Dynamic web pages on the other hand create content as requested [32].

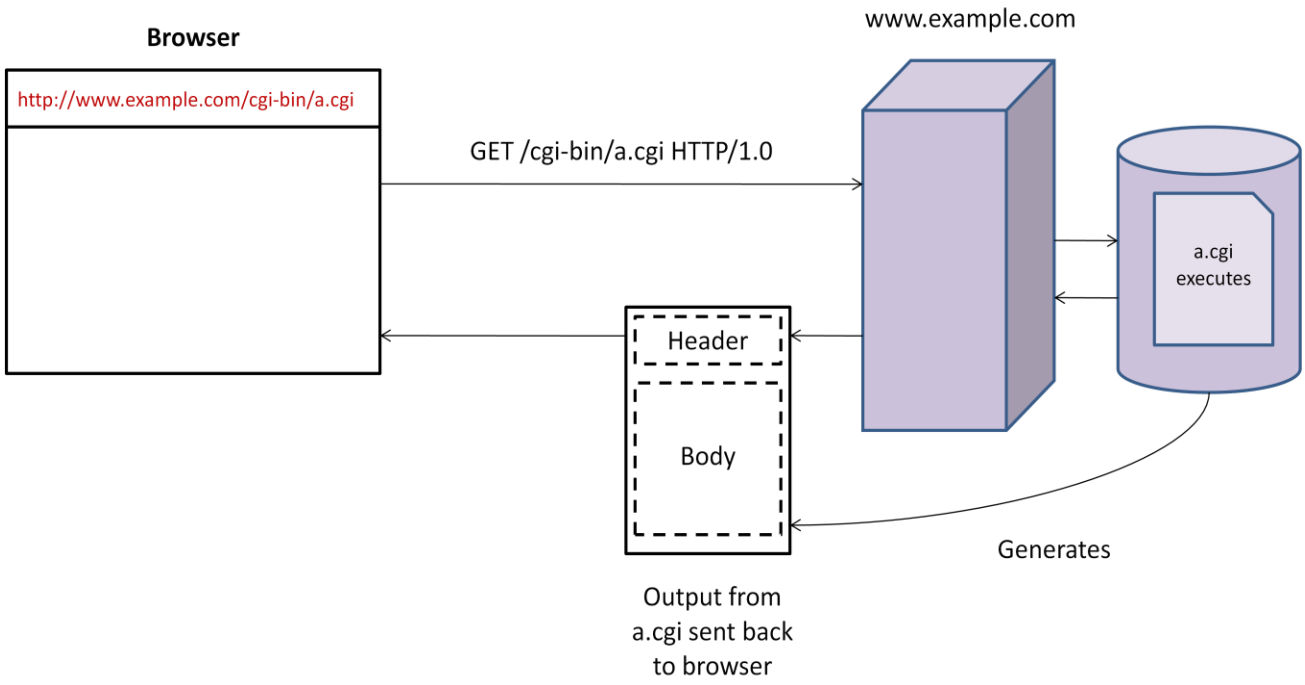


Figure 5.1: How CGI works [31]

Figure 5.1 shows what happens when a request for a CGI program is made. For example, when the URL *www.example.com/cgi-bin/a.cgi* is loaded into the web browser, the web server *www.example.com* is contacted [32]. The server receives a request like-

GET /cgi-bin/a.cgi HTTP/1.0

On receiving this request the server looks into the *cgi-bin* directory for the program *a.cgi* and executes it as a stand-alone program. The program generates output in the HTML format: a header, a blank line and the body [32].

5.1 Apache Configuration

The Apache HTTP server is an open-source web server software for modern operating systems including UNIX and Windows NT. The Apache HTTP server version 2.2 was installed on the Ubuntu 10.04 operating system for this project. The web pages created in this project were not

put on the internet and therefore cannot be accessed from anywhere outside the host PC. Therefore the server and the client (web browser) both reside on the same machine.

After installing Apache, its configuration files should be modified from their default values. First of all the *ServerAdmin* directive in *default* file (default is the name of the file) should be changed in the following way-

ServerAdmin you@youraddress.com

The location of the log file can be specified using the *CustomLog* directive in *default*. The *CustomLog* options can be used to specify the amount of information to be logged on every hit of the web server. This can be done in the following way-

CustomLog /var/log/apache2/access.log combined

The *combined* attribute used with *CustomLog* directive tells the server to log the maximum amount of information.

The user and group can be set in the *envvars* file in the following way-

export APACHE_RUN_USER=apache
export APACHE_RUN_GROUP=apache

The user and group can be apache or anything else. Care should be taken that Apache is not run as *root* because if it was cracked then the hacker has access to the entire operating system.

The *server-status* and *server-info* directives allow clients to find information about the server and the machine it is running on. The *server-status* directive can be found in *status.conf* and *server-info* directive can be found in *info.conf*.

The *.htaccess* file can be used to change the configuration of files on a per-directory basis. This file contains configuration directives which are applied to the directory in which the file is located and all of its subdirectories. The name of this configuration file is specified using the *AccessFileName* directive in the *apache2.conf* file. The file shall not necessarily be called *.htaccesss*, it can be called something else also like *.config* [33].

Whenever the server finds the *.htaccess* file in a directory, it needs to know which directives declared in the file can override the earlier configuration directives. This can be done using the *AllowOverride* directive. For example if this directive is set to none then all the configurations specified in the *.htaccess* file will be completely ignored [33].

The *order*, *allow* and *deny* directives are used to control access to certain parts of the directory tree in which the web pages are stored. The order of allow and deny directives is important as the last match takes precedence. Consider the following example [34]-

Order Allow, Deny

Allow from apache.org
Deny from foo.apache.org

In the example above, all requests from *foo.apache.org* will be rejected while requests from all other hosts from *apache.org* will be accepted. But if the order was reversed then all requests from *apache.org* will be accepted [34]. The matching rules followed when using *allow* and *deny* directives are summarized in table 5.1

Match	Allow, Deny result	Deny, Allow result
Match Allow only	Request allowed	Request allowed
Match Deny only	Request denied	Request denied
No Match	Default to second directive: Denied	Default to second directive: Allowed
Match both Allow and Deny	Final match controls: Denied	Final match controls: Allowed

Table 5.1: Matching rules followed when using Allow and Deny directives [34]

The *.htaccess* file can be used for password protection of web pages in a directory also. To accomplish this, first the *AllowOverride* directive should be set to *AuthConfig*. Then a file should be created to store the usernames and passwords. This file should be saved outside the directory tree in which the web pages are stored because in case of a server misconfiguration the password file will be accessible to the outside world [32]. The password file can be created in the following way-

htpasswd –bc xyz.passwords neo Anderson

The *–b* option means that the password is supplied in the command line and the *–c* option means that the file should be created. Here *neo* is the username, *Anderson* is the password and *xyz.passwords* is the file in which the passwords are stored [32]. Lastly, the *AuthType*, *AuthName*, *AuthUserFile*, *AuthGroupFile* and/or *Require* directives should be set in the *.htaccess* file for the configuration changes to take effect.

5.2 Using Perl to write CGI scripts

When using Perl to write CGI scripts, the following modules were used-

- Device::SerialPort- This module was used to read from and write data to a serial port in Unix.
- Time::gmtime- This module was used to display time on web pages.
- CGI ‘:standard’-This module provides methods to create dynamic web pages.
- CGI::Widget::Tabs-This module was used to create tabs on web pages.

One thing to note here is that when a serial port is accessed by a Perl script executed by a *super user* in Unix then there is no problem but when the Apache server running a CGI script tries to do the same thing, it is denied access. This is because the group to which the Apache server belongs does not have permissions to access the serial port. For example, in this project *ttyUSB0*

serial port is used to communicate with the active RFID reader. This port can be accessed by a user belonging to a *root* group or the *dialout* group only. Therefore the Apache user name should be added to the *dialout* group in order for the Apache server to be allowed access to the *ttyUSB0* port.

5.3 How GUI works

The GUI implemented in this project provides the user the capability to program both the active RFID reader and the tag. Fig. 5.2 shows the GUI used to program the reader. It can be used to issue one out of seven commands at a time to the reader. The commands that can be sent to the reader are-

- **Read Queue Length**: The reader will return the number of tag commands stored in its memory. The queue length should be less than or equal to thirty two.
- **Delete Queue Entry**: The user will provide a command handle and the reader will delete the corresponding tag command from its memory.
- **Read Queue Entry Command**: The user will provide the command handle and the reader will return the corresponding tag command.
- **Read Reader ID**: The reader will return its ID.
- **Flush Queue**: The reader will delete all the tag commands from its memory.
- **Set Reader Time**: The time on the internal clock of the reader will be set to the time specified by the user.
- **Query Reader Time**: The reader will return the time on its internal clock.

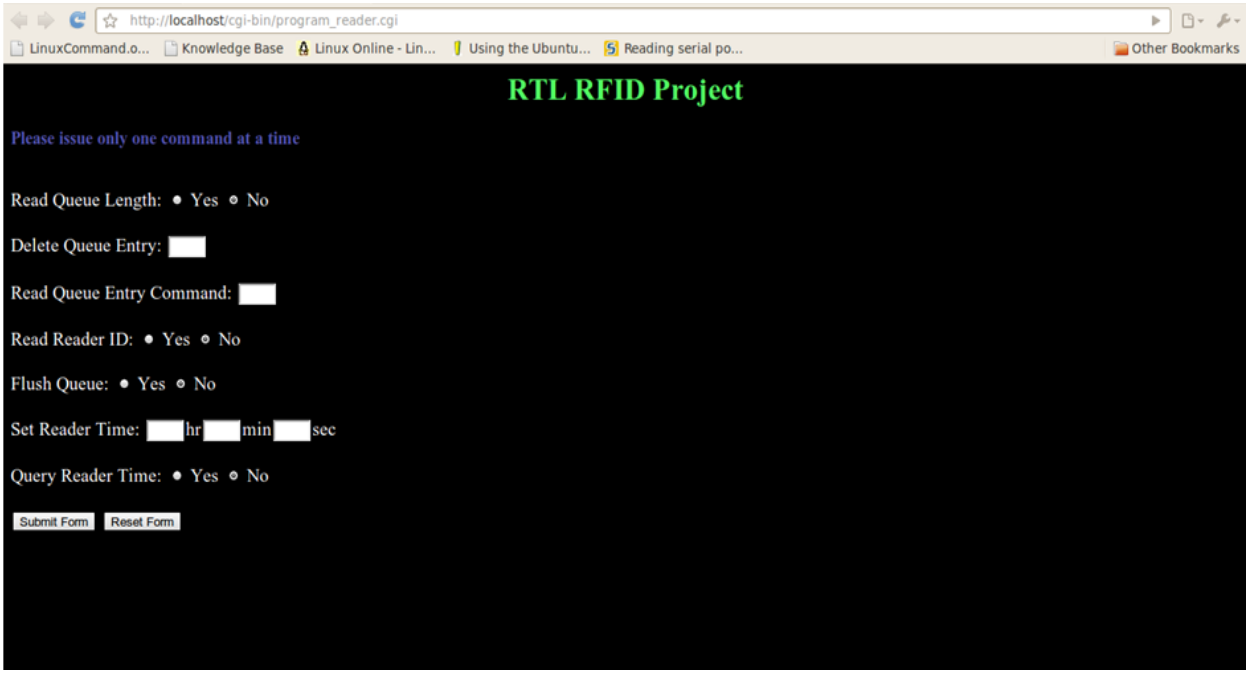


Figure 5.2: GUI to program the reader

To program the tag six different tabs are provided. Similar to programming the reader, the tag is also programmed by issuing commands to the reader, the only difference here being that multiple parameters can be specified in a single command. Figure 5.3 shows the GUI to program the network field parameters of the tag. The various parameters that can be modified in this field are described below-

- **Mode**: This parameter specifies the mode that will be affected by the command.
- **Set/Read Tx Interval**: This parameter is used to either set or read the value of the tag transmit interval. If the transmit interval is read, then its value will be returned only once- in the next packet transmitted by the tag after receiving the command.
- **Enable/Disable Reporting of Tag Tx Interval**: This parameter can be used to turn on or off the reporting of the transmit interval by the tag. If this parameter is enabled then the tag will report its transmit interval in every subsequent packet until this parameter is disabled by the user.
- **Set/Read Tx Power**: Similar to *Set/Read Tx Interval* but instead of transmit interval the output power of the tag is set or read.
- **Enable/Disable Reporting of Tag Tx Power**: Similar to *Enable/Disable Reporting of Tag Tx Interval* but instead of transmit interval the tag's output power reporting is enabled or disabled.
- **Change Tag ID**: This parameter can be used to change the Tag ID. The new tag ID will be set to the value specified by the user.

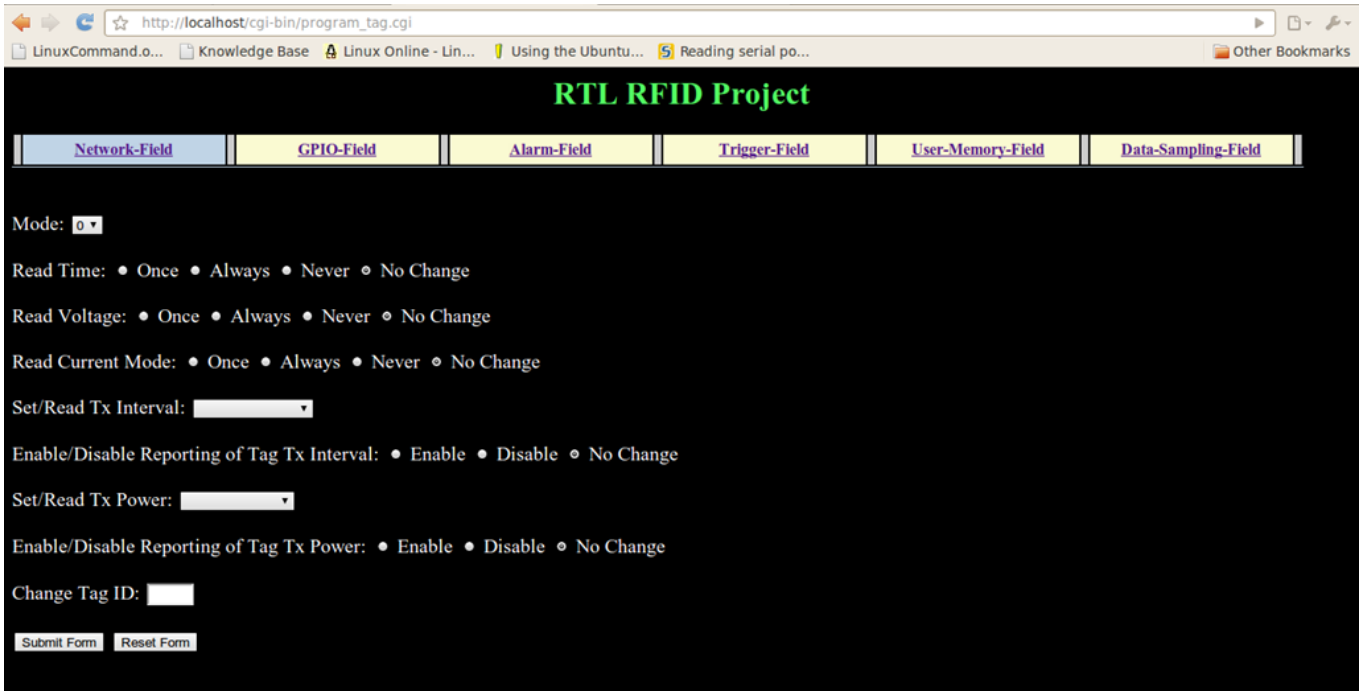


Figure 5.3: GUI to program the network field parameters of the tag

The GUI to program GPIO field parameters is shown in fig. 5.4. There are two digital and two analog pins that can be used to interface sensors with the tag. Currently, only one analog pin is used and a temperature sensor is attached to that pin. GUI gives the option of configuring the other pins also in case they are used in the future. The various GPIO field parameters that can be modified are-

- **Pins to be configured**: This parameter is used to specify which pins are to be configured with this command. All other pins will remain unaffected even if parameters pertaining to them are modified.
- **Digital Pin 1**: This parameter specifies whether digital pin 1 is an input or an output pin.
- **Digital Pin 2**: This parameter specifies whether digital pin 2 is an input or an output pin.
- **Digital Pin 1 (value)**: This parameter specifies the value of pin1 in the case that this pin is being used as an output pin.
- **Digital Pin 2 (value)**: This parameter specifies the value of pin2 in the case that this pin is being used as an output pin.
- **Temperature Sensor (Reference Voltage)**: This parameter specifies the reference voltage for the temperature sensor.
- **Analog Sensor (Reference Voltage)**: This parameter specifies the reference voltage for the analog sensor.
- **Temperature Sensor**: This parameter can be used to read and record the value of the temperature sensor.
- **Analog Sensor**: This parameter can be used to read and record the value of the analog sensor.
- **Accelerometer:** This parameter can be used to read and record the value of the accelerometer module.

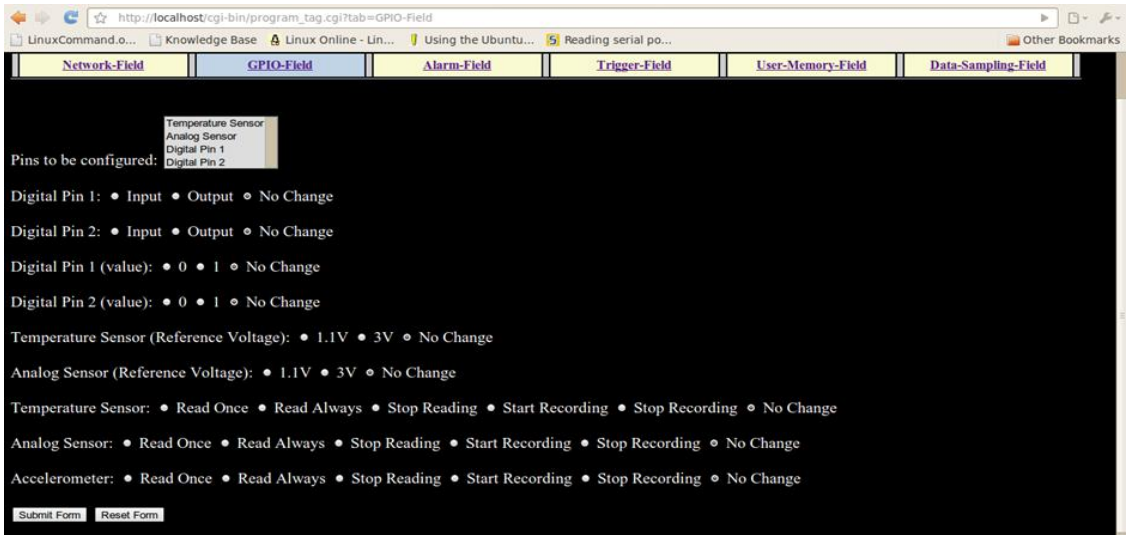


Figure 5.4: GUI to program the GPIO field parameters of the tag

The GUI to program alarm field parameters of the tag is shown in fig. 5.5. An alarm can be specified for each mode, thus there can be a maximum of eight alarms enabled at a time. If the time period for two alarms overlap, then the alarm specified for the higher mode number will take precedence. The tag's internal clock does not keep track of the date; it is reset every 24 hours. Thus the alarms will recur every day. For example, if an alarm has been set to enter *Mode2* at 8A.M. then the tag will enter *Mode2* at 8 A.M. every day. The alarm field parameters that can be configured are-

- **Mode**: This parameter specifies the mode the tag will enter when the alarm being set is activated.
- **Read Alarm Start Time and Expiration Time**: This parameter is used to read the start time and expiration time of the alarm corresponding to the mode specified by the *Mode* parameter above. These times will be reported only in the next packet that the tag transmits after receiving this command.
- **Start Time**: This parameter specifies the start time for the alarm being set.
- **End Time**: This parameter specifies the time when the alarm being set is deactivated.
- **Enable Alarm**: This parameter is used to enable the alarm corresponding to the mode specified by the *Mode* parameter. Any alarm will not be enabled until it has been explicitly enabled by setting this parameter.

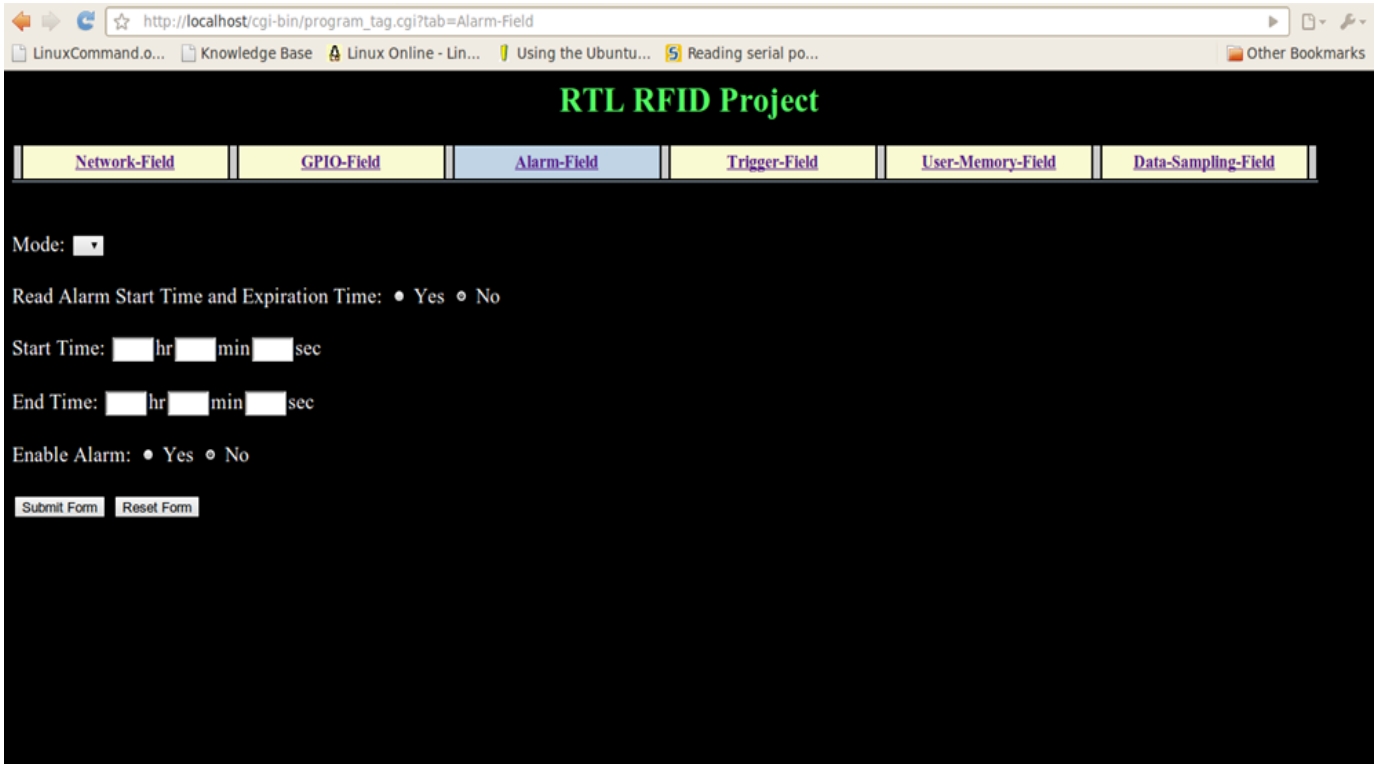


Figure 5.5: GUI to program alarm field parameters of the tag

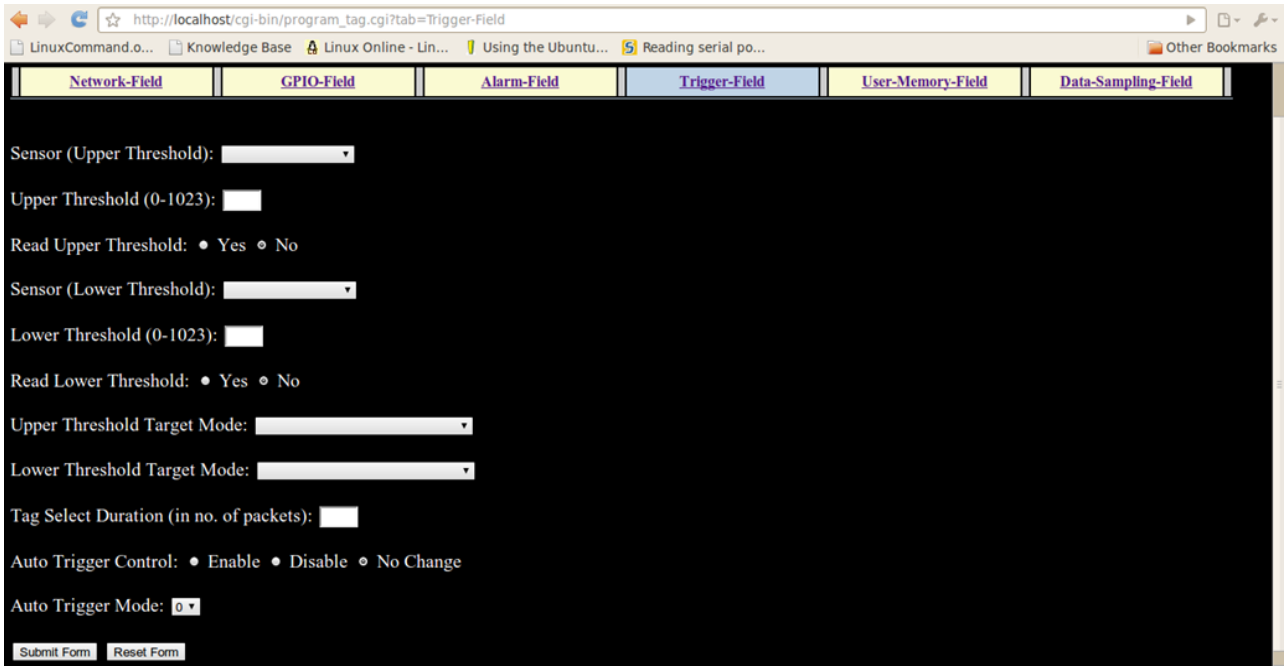


Figure 5.6: GUI to program trigger field parameters of the tag

Figure 5.6 shows the GUI to program the trigger field parameters of the tag. The various parameters that can be configured in this field are-

- **Sensor (Upper Threshold)**: This parameter specifies the sensor whose value will be compared with the upper threshold value. As already stated, there is a possibility of having two digital and two analog sensors on the tag, but currently only one analog sensor (temperature sensor) is incorporated on the tag.
- **Upper Threshold**: This parameter specifies the upper threshold value. This value should be between 0 and 1023. The default value of this parameter is 1023.
- **Read Upper Threshold**: This parameter can be used to read the upper threshold value.
- **Sensor (Lower Threshold)**: This parameter specifies the sensor whose value will be compared with the lower threshold value.
- **Lower Threshold**: This parameter specifies the lower threshold value. This value should be between 0 and 1023. The default value of this parameter is 0.
- **Read Lower Threshold**: This parameter can be used to read the lower threshold value.
- **Upper Threshold Target Mode**: This parameter specifies the mode the tag should enter when the value of the sensor specified by the *Sensor (Upper Threshold)* parameter goes above the upper threshold value.
- **Lower Threshold Target Mode**: This parameter specifies the mode the tag should enter when the value of the sensor specified by the *Sensor (Lower Threshold)* parameter goes below the lower threshold value.

- **Tag Select Duration**: When this parameter is specified by the user then the tag will transmit every half a second. This behavior will continue till the number of packets transmitted by the tag is equal to the value specified by the user for this parameter. After that the tag will return to the mode it was previously in.
- **Auto Trigger Control**: This parameter can be used to stop the tag from changing its mode based upon the alarm field and trigger field parameters. If *Auto Trigger Control* parameter is disabled then the tag cannot change its mode and will remain in its current mode indefinitely unless this parameter is re-enabled.
- **Auto Trigger Mode**: This parameter specifies the mode that the tag should enter after auto triggering is turned off.

Figure 5.7: GUI to program user memory field parameters of the tag

The GUI to program user memory field parameters of the tag is shown in fig. 5.7. The various parameters that can be configured in this field are-

- **Read/Write**: This parameter specifies whether data is to be written to or read from the user memory (EEPROM).
- **Display data in all packets**: This parameter specifies whether the data read from the user memory is reported in all packets transmitted by the tag or not.
- **Display start address in the tag data packet**: This packet specifies whether the start address of the data read from the user memory is included when reporting the user memory data or not.
- **Start Address**: This parameter specifies the start address in the user memory from which the data has to be read or to which data has to be written. This address can only be between 0 and 1023.

- **Data Size**: This parameter specifies the number of bytes of data to be read from the user memory. The maximum value for this parameter can be 16.
- **Data**: This parameter specifies the data to be written to the user memory. The data should be in hexadecimal format and less than or equal to 16 bytes.

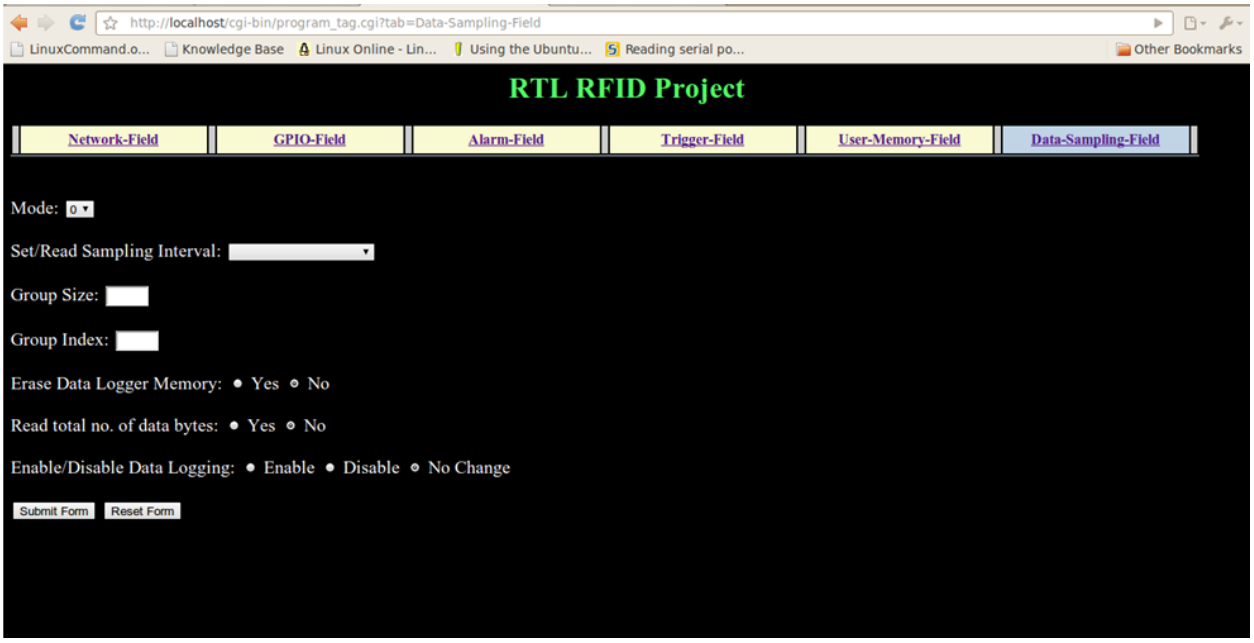


Figure 5.8: GUI to program data sampling field parameters of the tag

Figure 5.8 shows the GUI to program the data sampling field parameters of the tag. The various parameters that can be configured in this field are-

- **Mode**: This parameter specifies the mode which will be affected by the command.
- **Set/Read Sampling Interval**: This parameter is used to either set or read the value of the sampling interval. If the sampling interval is read, then its value will be returned only once- in the next packet transmitted by the tag after receiving the command.
- **Group Size** and **Group Index**: These parameters are used to read data from the data logger memory (flash memory). Consider the following example-

  Total no. of bytes in data logger memory-70
  Group Size-3
  Group index-0

  Group Size parameter is used to partition the memory into groups. In the example above, the memory is partitioned into 3 groups, first one from byte 1 to byte 23 (as [70/3]=23 where [x] denotes the integer part of x), second one from byte 24 to 46 and third one from byte 47 to 70. *Group Index* parameter denotes which group has to be reported. In the above example the first group (from byte 1 to byte 23) is reported. The maximum number of data logger memory bytes reported in a single packet is 16. Therefore the data

from first group will be transmitted in two data packets sent one after the other without any delay between them; the first packet will contain data from byte 1 to byte 16 while the second packet will contain data from byte 17 to byte 23.

- **Erase Data Logger Memory**: This parameter can be used to erase all the data in the data logger memory.
- **Read total no. of data bytes**: This parameter can be used to read the total no. of data bytes stored in the data logger memory.
- **Enable/Disable Data Logging**: If this parameter is enabled then the tag will continue to record data in its data logger memory otherwise data logging will be disabled by the tag.

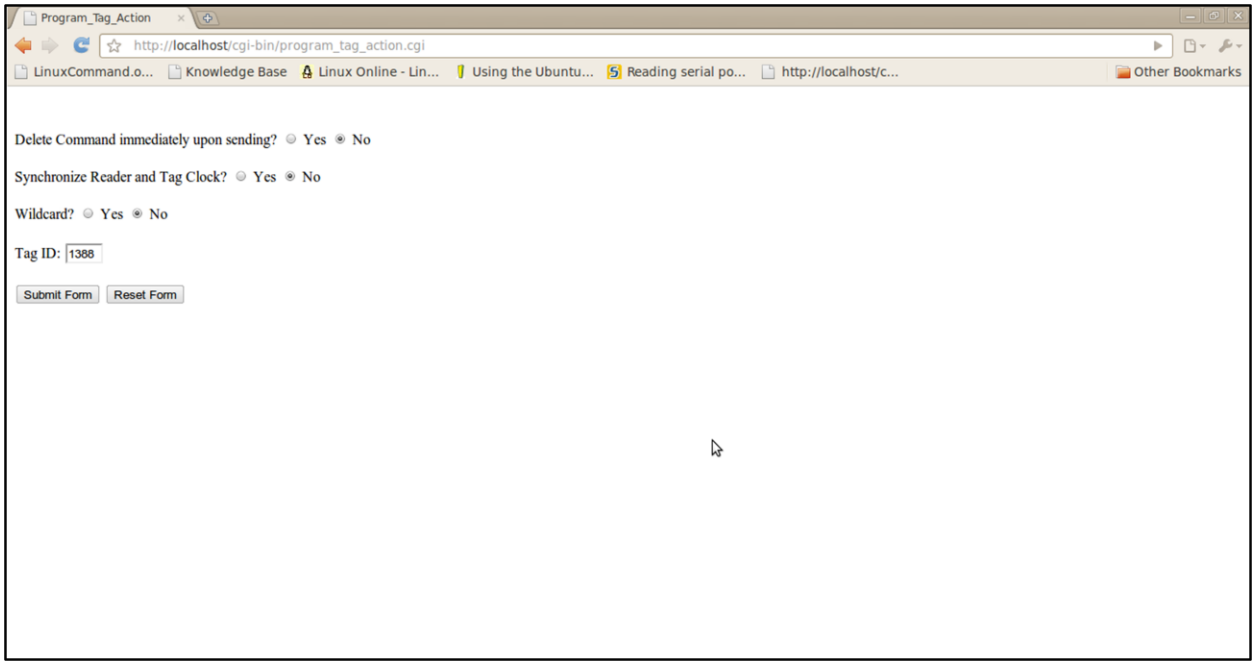


Figure 5.9: GUI to give instructions to the reader while sending a tag command

Figure 5.9 shows the GUI to give instructions to the reader while sending a tag command. The parameters that can be configured here are-

- **Delete command immediately upon sending**: The reader will continue to send a command to the tag till the time it does not receive an acknowledgement back from the tag. But if this parameter is set then the command will be deleted from the reader memory as soon as the command is sent to the tag.
- **Synchronize reader and tag clock**: This parameter can be used to tell the reader to synchronize the tag clock with its own clock. The reader does this by transmitting the time on its internal clock with the command packet sent to the tag so that the tag can synchronize its clock with the reader clock.

- **Wildcard**: This parameter is used to tell the reader to transmit this command to the first tag that makes contact with it regardless of the tag ID. The *Tag ID* parameter should still be specified in the command although it will be of no use.
- **Tag ID**: This parameter gives the identity of the tag to which the command has to be sent by the reader.

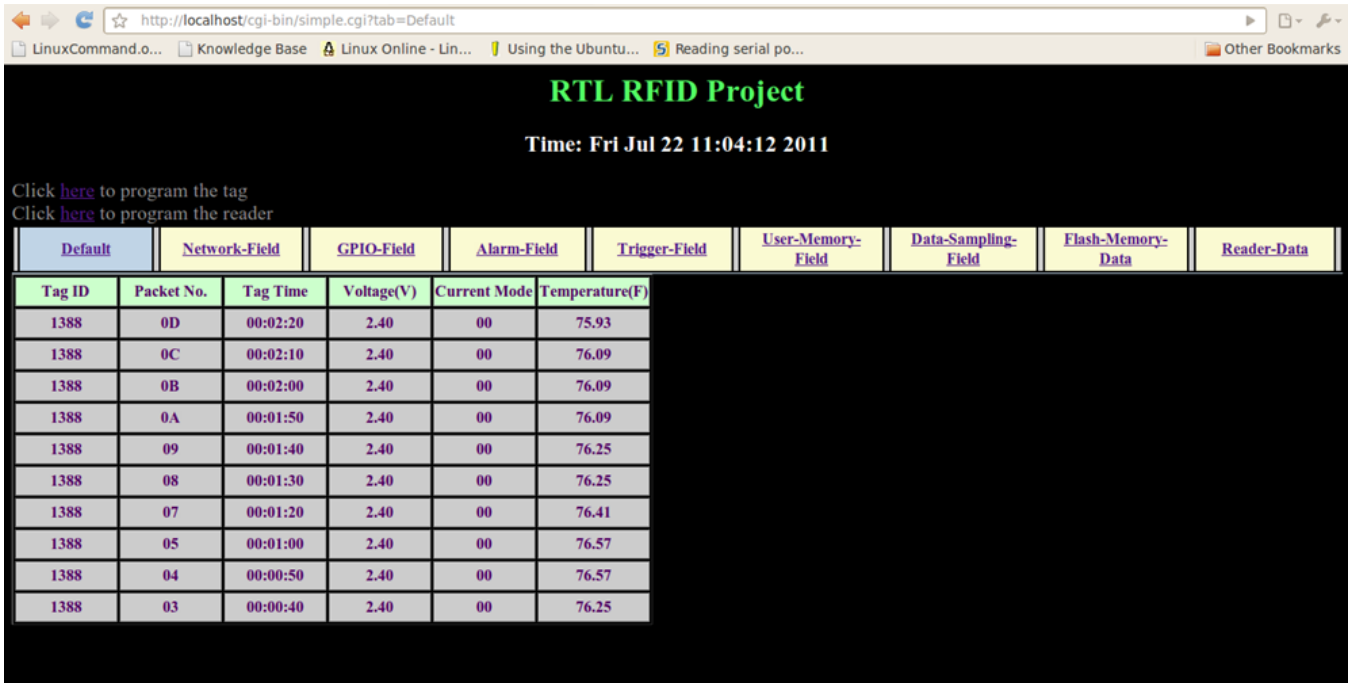


Figure 5.10: GUI to display the data received from the tag and the reader

Figure 5.10 shows the GUI which displays the data received from the tag and the reader. This GUI will be used in the next section to illustrate the effect of several commands on the tag and the reader.

6 Results

6.1 Electrical Characteristics

6.1.1 Current Consumption

The tag mainly has four modes- sleep, active, transmit and receive. The tag is mostly in the sleep mode but goes into the active mode every half a second to check whether it has to transmit or not. If it doesn't have to transmit then it goes back into the sleep mode and remains in that mode for another half a second and so on.

Before transmitting data, the tag goes into the receive mode to sense the channel and if the channel is clear then it goes into the transmit mode to transmit the data. After the transmission is over it goes back into the receive mode and remains there for a brief period to receive commands from the reader. After that it goes back into the sleep mode or transmits an acknowledgement back to the reader and then goes into the sleep mode depending on whether or not it receives a command from the reader. The acknowledgement will only be transmitted if it is explicitly asked by the reader.

The tag is in the active mode if it is performing some computations. Although, when the tag is transmitting or receiving all the three major components of the tag-ATmega328P, CC2520 and CC2591 are in the active state, but we refer the tag to be in the transmit mode or the receive mode as per the RF behavior of the tag. The tag is said to be in the active mode only when it's not in the sleep mode and when it's not using its RF functionalities. Another way to look at this is to say that the tag is in the active mode whenever it is not in any of the other three modes.

Figure 6.1: Sleep mode current in the reprogrammed ZT-500 tag

Mode	Current consumed
Sleep	< 2 µA
Receive	< 30 mA
Transmit	< 160 mA

Table 6.1 : Current consumption in the ZT-500 Tag [17]

The sleep mode current in the reprogrammed tag was measured to be 1.7 µA as shown in fig.6.1. The ZT-500 tag consumes less than 2µA of current in sleep mode as shown in table 6.1 which is consistent with the sleep mode current consumption of the reprogrammed tag. Table 6.2 shows the current consumption of the major hardware components of the ZT-500 tag in the sleep mode (ATmega328P in power-save mode and CC2520+CC2591 combination in LPM2 mode). If the current consumption in these components is added up then the total will be around 2µA, which is again consistent with the values obtained from the reprogrammed tag.

Component	Condition	Typical	Max.
ATmega328P [20]	Power-save mode, 32KHz TOSC enabled, V_{CC}=3V	0.9 µA	2.6 µA
ATmega328P[20]	Active, 4MHz, V_{CC}=3V	1.2 mA	3.5 mA
ATmega328P[20]	Active, 8MHz, V_{CC}=5V	4.0 mA	12.0 mA
CC2520+CC2591[16]	LPM2 mode	< 1 µA	
CC2520+CC2591[16]	Receive current, Wait for sync, -90 dBm input level	26 mA	
CC2520+CC2591[16]	Receive current, Wait for sync, -50 dBm input level	23 mA	

Table 6.2: Current consumption in the major hardware components of the ZT-500 tag

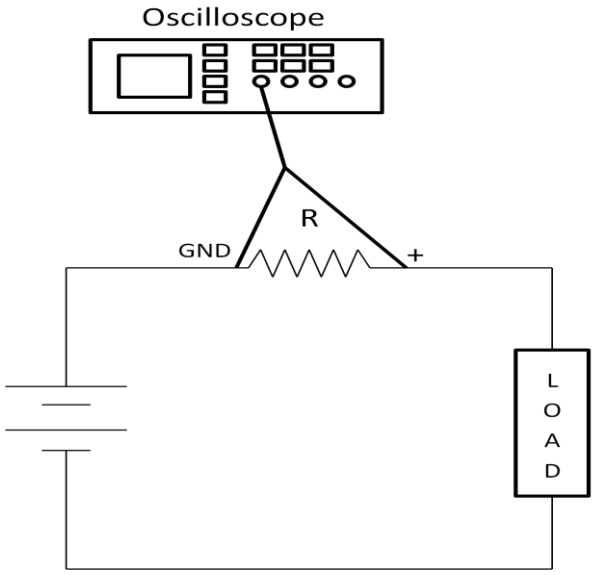


Figure 6.2: How current was measured in the active, receive and transmit modes in the reprogrammed ZT-500 tag

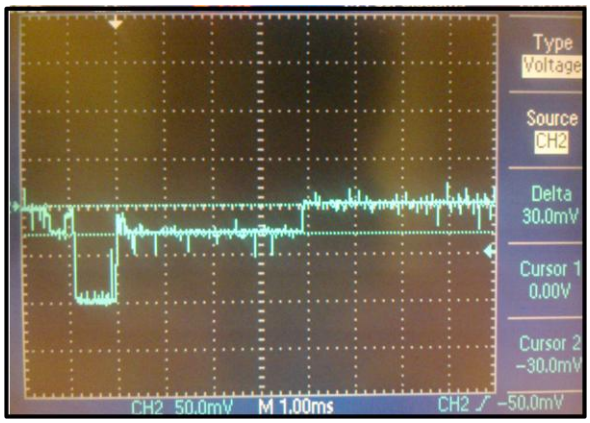


Figure 6.3: Current consumption in the receive mode in the reprogrammed ZT-500 tag

To measure the current consumption in the modes other than the sleep mode, a resistor in combination with the oscilloscope was used. An ammeter was not used to measure the current in the other modes as these modes exist for a very brief period only and then the tag goes back into the sleep mode. An ammeter is incapable of showing the current changes in milliseconds. The oscilloscope in addition to measuring the current in these modes will also provide the time for which these modes exist. Figure 6.2 shows the way in which the current was measured in these modes in the reprogrammed ZT-500 tag.

The current consumed in the receive mode in the ZT-500 tag is shown in fig. 6.3. The value of resistor *R* in fig. 6.2 was equal to 1Ω when the setup was used to measure the receive mode and transmit mode current in the reprogrammed tag. Therefore from fig. 6.3 the receive mode current will be equal to 30mA which is again in agreement with the receive mode current in the ZT-500 tag as shown in table 6.1 and the receive current in CC2520+CC2591 combination circuit as shown in table 6.2. The voltage in fig. 6.3 is negative because the ground probe of the oscilloscope was connected towards the V_{CC} side of the circuit shown in fig. 6.2 for convenience. The transmit mode current in the reprogrammed tag for different power levels is shown in fig. 6.4 and summarized in table 6.3. The transmit mode current values obtained are quite different from the values shown in table 4.3. The reason for this might be that the values reported in table 4.3 were measured on the CC2520-CC2591 EM reference design with a 50Ω load at +3.0V, +25ºC and f_C=2440MHz.

TXPOWER	Current [mA]
0xF9	104
0xF0	96
0xA0	84
0x2C	72
0x03	78
0x01	76

Table 6.3: Current consumption in the transmit mode in the reprogrammed ZT-500 tag

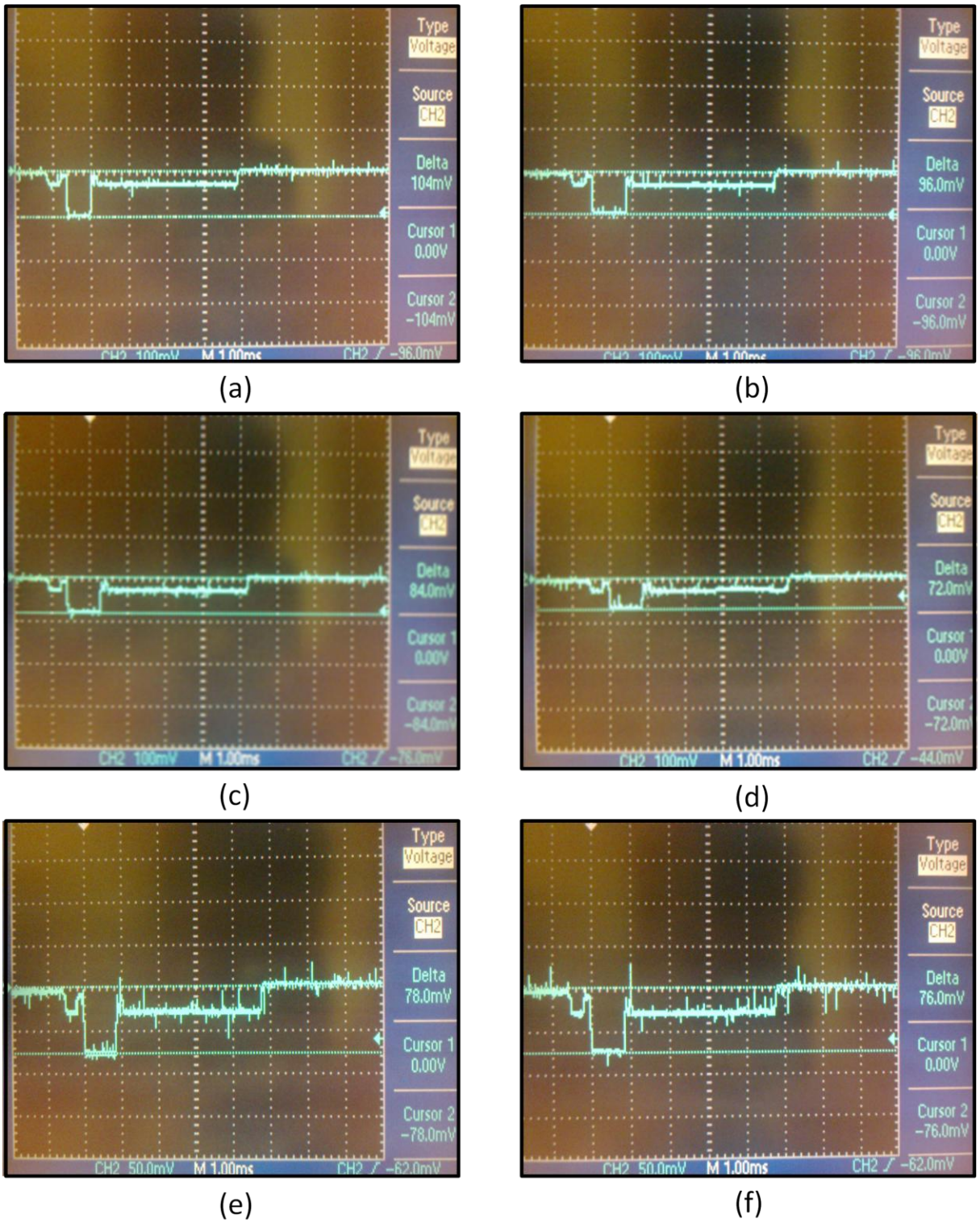


Figure 6.4: Current consumption in the transmit mode in the reprogrammed ZT-500 tag (a) Tx Power:17dBm (b) Tx Power: 16 dBm (c) Tx Power: 14 dBm (d) Tx Power: 11 dBm (e) Tx Power: -1 dBm (f) Tx Power: -8 dBm

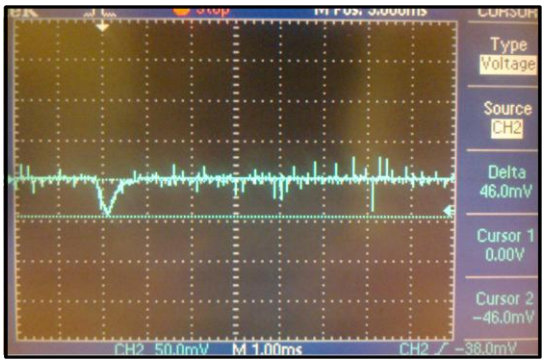


Figure 6.5: Current consumption in the active mode in the reprogrammed ZT-500 tag

The current consumed in the active mode in the ZT-500 tag is shown in fig. 6.5. The value of resistor *R* in fig. 6.2 was equal to 20Ω when the setup was used to measure the active mode current in the reprogrammed tag. Therefore from fig. 6.5 the active mode current will be equal to 2.3mA. This measurement only indicates the current consumption in the ATmega328P microcontroller when it's in the active state, it does not include the current consumed by CC2520 and CC2591 when they are in the active state. Table 6.2 states the current consumption in ATmega328P in the active state for different values of clock frequency and V_{CC}. The clock frequency and V_{CC} used for ATmega328P in this project were 8MHz and 3V respectively.

6.1.2 Duration of Different Modes

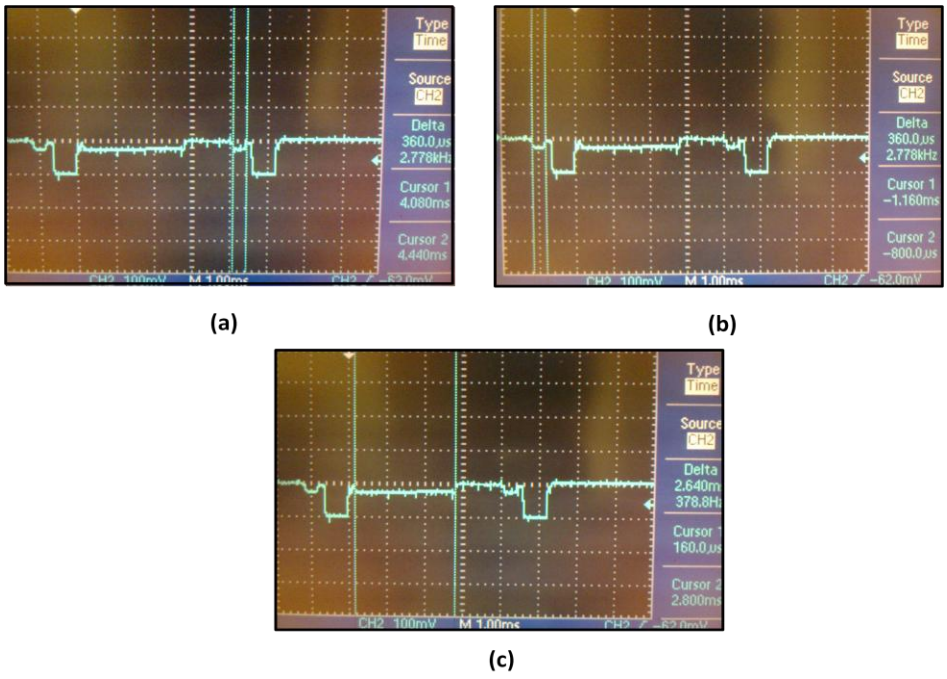


Figure 6.6: Duration of receive mode (a) before transmitting acknowledgement (b) before transmitting data (c) after transmitting data

Figure 6.6 shows the duration of receive mode. The receiver is turned on before transmitting to sense the channel. The duration of time for which the receiver is turned on comes out to be 360µs. The receiver is also turned on after transmitting data to receive any commands from the reader. This duration should be long enough to give the reader sufficient time to find out which command has to be sent to the tag and then to transmit that command. This time comes out to be around 2.64ms.

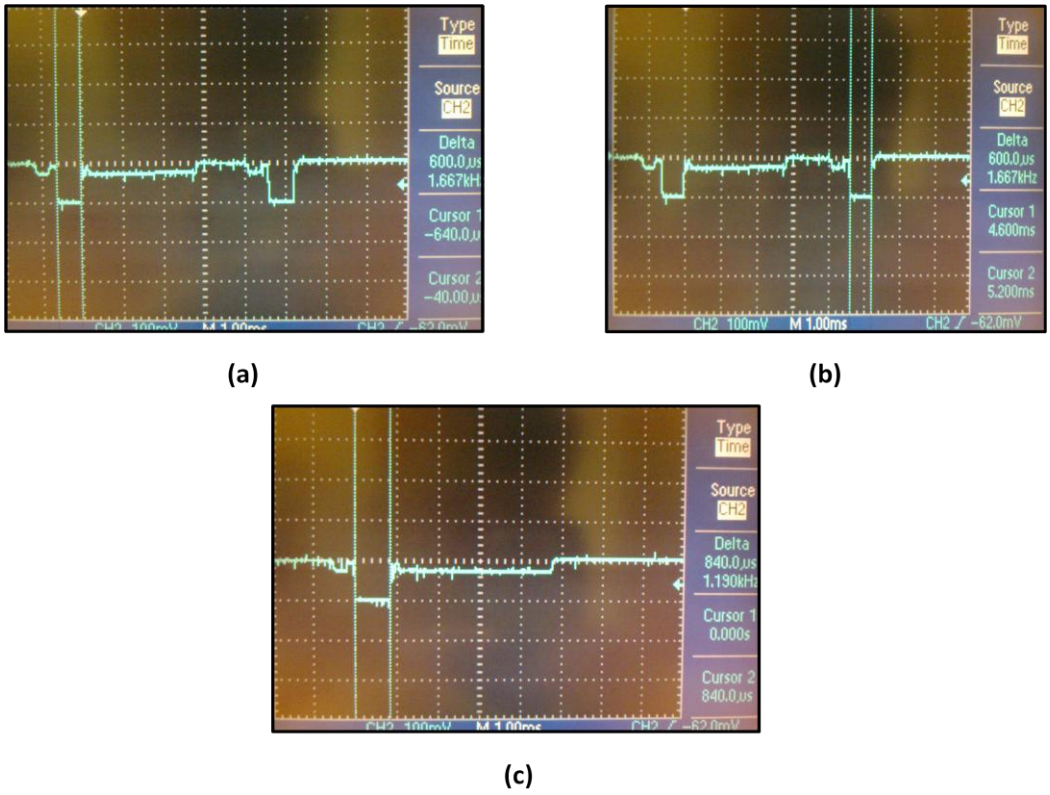


Figure 6.7: Duration of transmit mode (a) standard data packet is transmitted (b) acknowledgement is transmitted (c) longer data packet is transmitted

The duration of transmit mode is shown in fig. 6.7. The tag will transmit its tag ID (2 bytes), protocol version & packet type byte and a PCH byte (refer to [18]) unless otherwise asked by the reader. When the tag is transmitting an acknowledgement then it only transmits its tag ID, protocol version & packet type byte and a response code byte (again refer to [18]). Thus the *command payload* as shown in fig. 4.4 is 4 bytes long for both the standard data packet and the acknowledgement. The other bytes in the transmitted packet are as follows-

- Preamble sequence: 4 bytes
- Start of Frame Delimiter: 1 byte
- Frame Length: 1 byte
- Frame Control: 2 bytes
- Sequence Number: 1 byte

- Address Information: 4 bytes
- Frame Check Sequence: 2bytes

Thus the total number of bytes transmitted in both the cases will be equal to 19. The data rate of the CC2520 transceiver is 250 kbps. Therefore to transmit 19 bytes the time required will be equal to-

$$\frac{19\times 8}{250\times 10^3} = 608\mu s$$

Fig. 6.7(a) and fig. 6.7(b) show this time to be equal to 600µs. This is because the sampling rate of the oscilloscope was 25 kHz and therefore it cannot represent a value like 608µs. Fig. 6.7(c) shows the transmit mode duration when a longer packet of 27 bytes (command payload 12 bytes long) was transmitted. Again, the time required to transmit 27 bytes of data can be calculated as follows-

$$\frac{27\times 8}{250\times 10^3} = 864\mu s$$

The duration of transmit mode in fig. 6.7(c) is shown to be 840µs. The difference in the theoretical value and the measured value can again be attributed to the low sampling rate of the oscilloscope.

On exiting the receive mode after receiving a command from the reader, the tag enters the active mode and not the sleep mode and remains in that mode until the receiver is again turned on right before the transmission of the acknowledgement takes place.

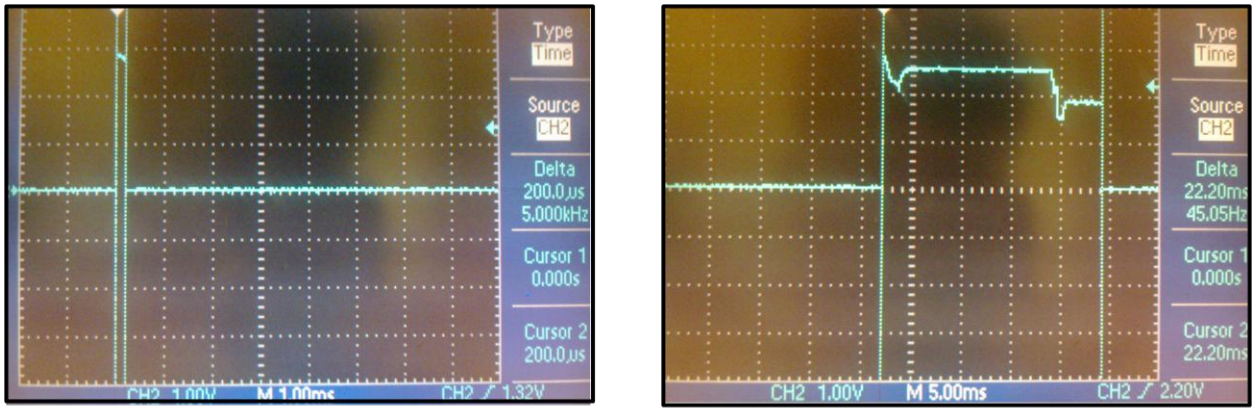


Figure 6.8: Measuring duration of active mode using LED (a) when no transmission takes place (b) when data is transmitted

To find out the duration of active mode, a LED was turned on every time the tag came out of sleep mode and the LED was turned off as soon as the tag went back into the sleep mode. Fig.6.8

shows the duration of time for which the LED was turned on. When no transmission takes place then the duration of active mode comes out to be 200µs as shown in fig. 6.8(a). When data is transmitted, the duration of active mode comes out to be around 22.20ms as shown in fig. 6.8(b). But it should be noted that in the last few milliseconds of voltage signal shown in fig. 6.8(a), receive mode and transmit mode are also activated. Therefore the duration of active mode will be around 20ms.

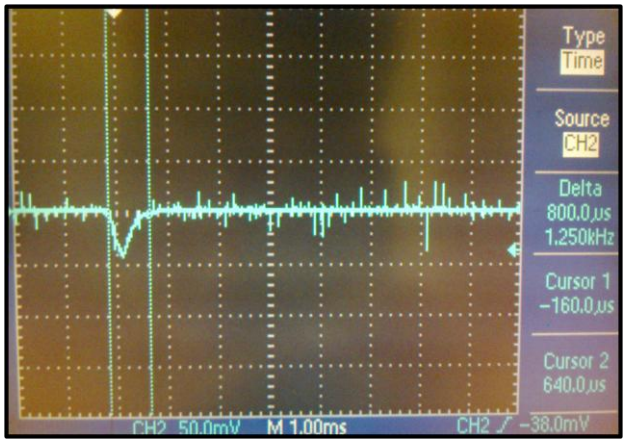


Figure 6.9: Measuring the active mode duration using the active mode voltage signal

Fig. 6.9 shows another way in which the active mode duration can be measured. If we use the voltage signal obtained in fig. 6.5 and measure the duration for which the voltage is non-zero then the duration of active mode comes out to be 800µs. The difference between the active mode durations obtained from fig. 6.9 and fig. 6.8(a) may be because of the time it takes for the microcontroller to ramp the current up and down. The LED will also have its own turn on and turn off time which can also contribute to the difference in the two active mode durations obtained. Thus, it makes more sense to assume the active mode duration obtained from fig. 6.9 to be more accurate.

6.1.3 Energy consumption in the Tag

Table 6.4 lists all the parameters that will affect the lifetime of the tag. The energy required for one transmission will be -

$$\begin{aligned}
&\approx \left(T_{AM2} \times I_{AM}\right) + \left(T_{RM1} \times I_{RM}\right) + \left(T_{TM} \times I_{TM}\right) + \left(T_{RM2} \times I_{RM}\right) \\
&\approx \left(20ms \times 2.5mA\right) + \left(400\mu s \times 30mA\right) + \left(650\mu s \times 104mA\right) + \left(3ms \times 30mA\right) \\
&\approx \left(50.0\mu As\right) + \left(12.0\mu As\right) + \left(67.6\mu As\right) + \left(90\mu As\right) \\
&\approx 219.6\mu As \approx 220\mu As
\end{aligned}$$

The tag wakes up every half a second and checks whether it has to transmit or not. Therefore the energy consumed every half a second will be-

$$\approx (0.5s \times I_{SM}) + (T_{AM1} \times I_{AM})$$
$$\approx (0.5s \times 2\mu A) + (1ms \times 2.5mA)$$
$$\approx (1.0\mu As) + (2.5\mu As)$$
$$\approx 3.5\mu As$$

Parameter	Value	Symbol
Battery Capacity	Two Duracell Alkaline AA batteries: Capacity=2900x2=5800 mAh [22]	E
Sleep Mode Current Consumption	2µA(approx.)	I_{SM}
Active Mode Current Consumption	2.5mA(approx.)	I_{AM}
Receive Mode Current Consumption	30mA	I_{RM}
Transmit Mode Current Consumption	104mA(Tx Power:17dBm)	I_{TM}
Active Mode Duration (when no packet is transmitted)	1ms(approx.)	T_{AM1}
Active Mode Duration (when transmission takes place)	20ms(approx.)	T_{AM2}
Receive Mode Duration (before transmission takes place)	400µs(approx.)	T_{RM1}
Receive Mode Duration (after transmission takes place)	3ms(approx.)	T_{RM2}
Transmit Mode Duration	650 µs(approx.)	T_{TM}

Table 6.4: Parameters affecting the lifetime of the reprogrammed ZT-500 tag

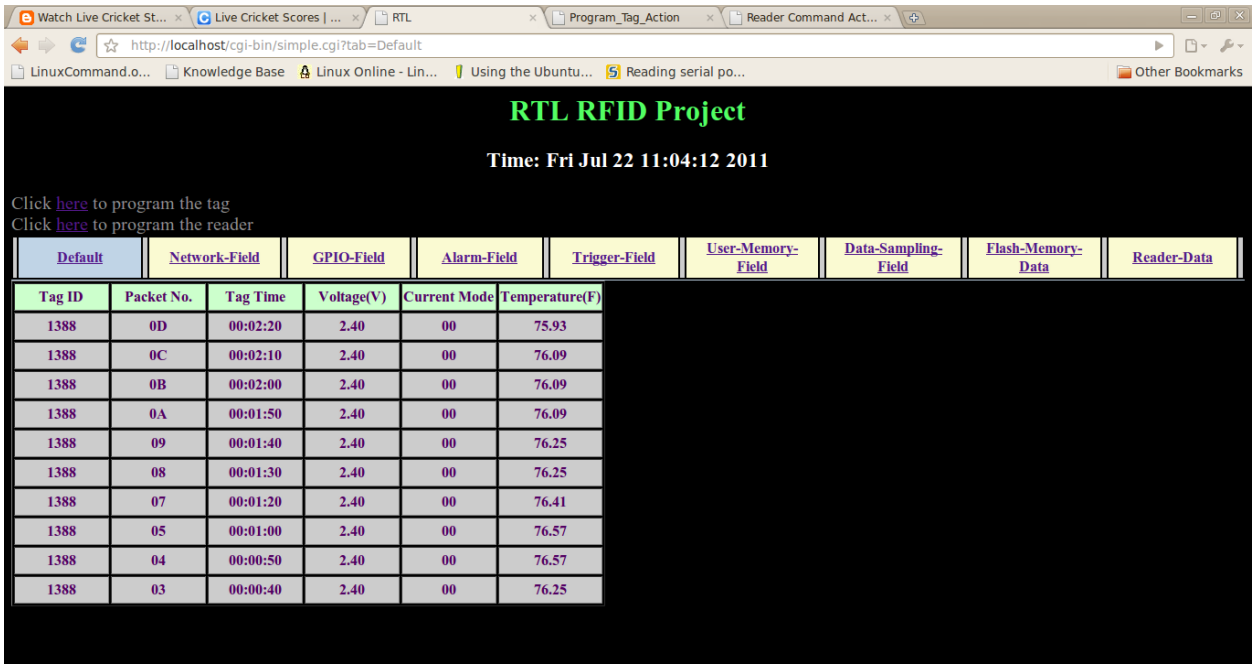
According to [17] the tag will last for 1 year when the transmit interval used is 1 minute, the battery capacity is 1550mAh and no additional sensors are attached to the tag. Therefore the tags used in this project can have an expected lifetime of around 3-4 years because of their greater battery capacity when no additional sensors are attached to them and the same transmit interval of 1 minute is used.

6.2 Testing the RFID System

In this sub-section, the RFID system developed in this project is tested. In this system the tag can be configured over the air by the reader. The user sends commands meant for a specific tag to the reader via a laptop and when the concerned tag contacts the reader, the reader immediately sends the command to that tag.

In this sub-section, the tag will be shown to be reporting some important parameters by default in every packet. This is done to make it easier to demonstrate the functionality of the tag. Otherwise

the tag only reports its ID in every transmitted packet unless explicitly asked by the user. Fig.6.10 shows the tag reporting the *tag ID, packet number, time on the tag clock, voltage of the battery on board the tag, the current mode* and the *temperature* fields.



Tag ID	Packet No.	Tag Time	Voltage(V)	Current Mode	Temperature(F)
1388	0D	00:02:20	2.40	00	75.93
1388	0C	00:02:10	2.40	00	76.09
1388	0B	00:02:00	2.40	00	76.09
1388	0A	00:01:50	2.40	00	76.09
1388	09	00:01:40	2.40	00	76.25
1388	08	00:01:30	2.40	00	76.25
1388	07	00:01:20	2.40	00	76.41
1388	05	00:01:00	2.40	00	76.57
1388	04	00:00:50	2.40	00	76.57
1388	03	00:00:40	2.40	00	76.25

Figure 6.10: GUI showing the tag reporting some of its important fields

Figure 6.11: GUI showing the time on the reader's internal clock

Values of the parameters specific to the reader can also be obtained by sending a command to the reader. Fig. 6.11 shows the reader sending back the time on its clock to the host computer.

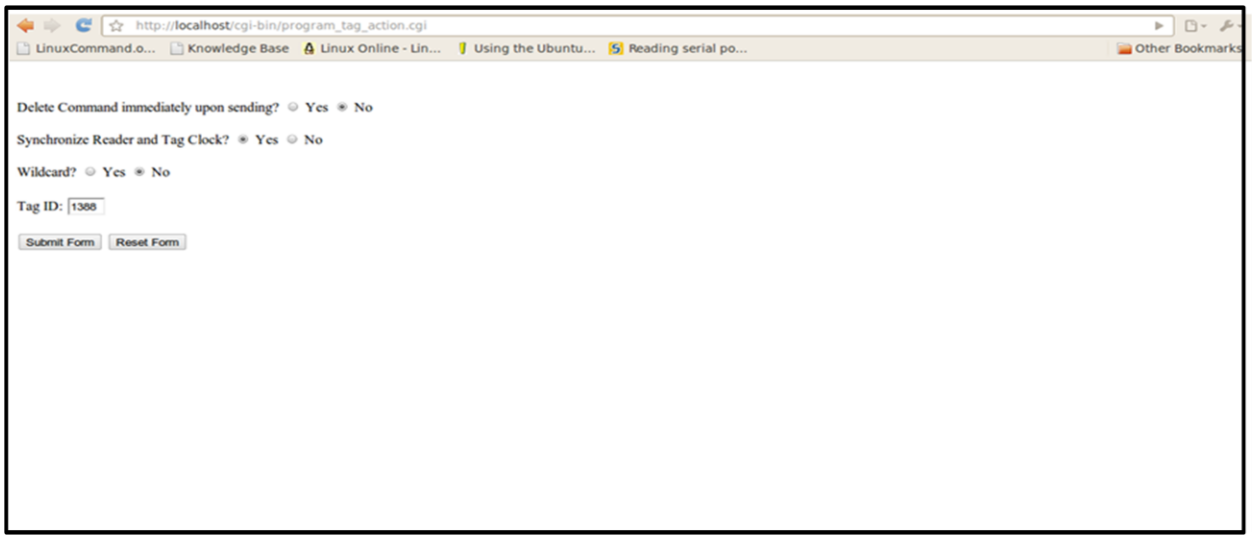


Figure 6.12: GUI to synchronize the time between the reader and the tag

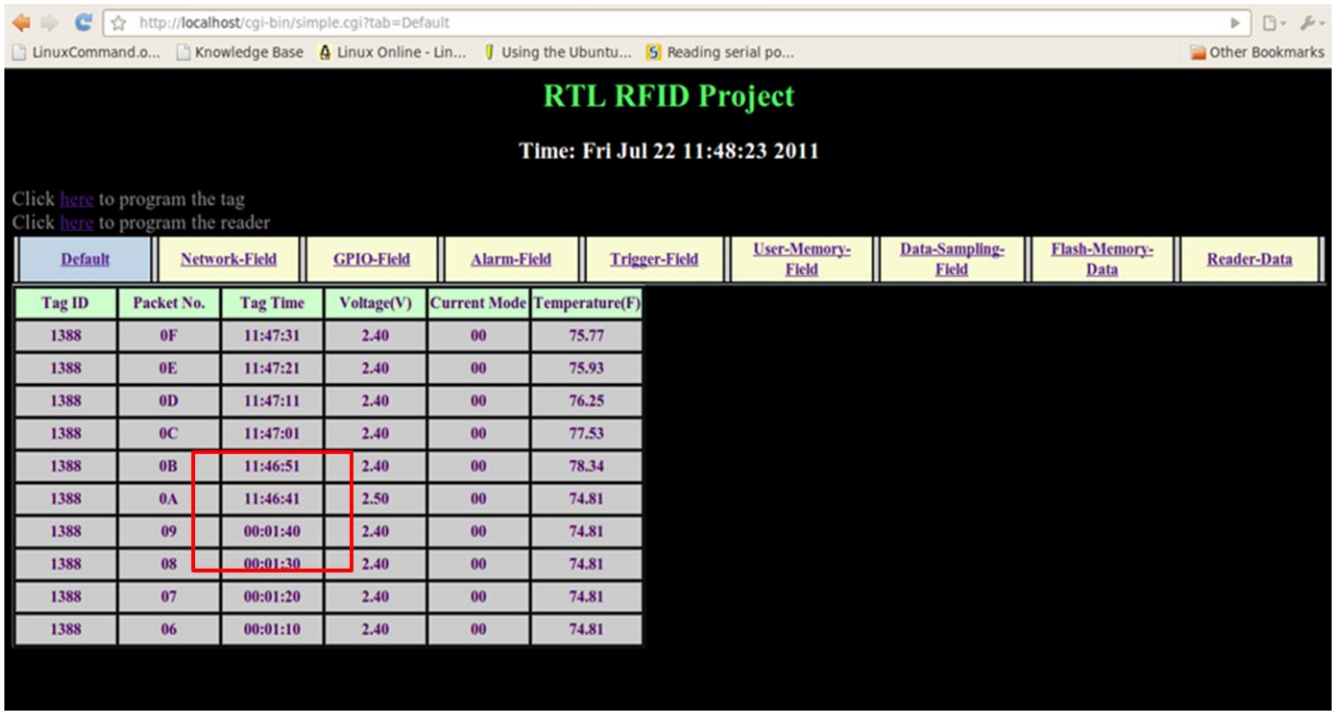


Figure 6.13: GUI showing the tag synchronizing its time with the reader

The time on the reader and tag clock can be synchronized by using the GUI shown in fig. 6.12. Fig. 6.13 shows the tag synchronizing its time with the reader.

Fig. 6.14 shows the tag reporting all its *Network Field* parameters. The tag can be configured to use different transmit interval and transmit power in different modes. Therefore, the *Mode Reported* parameter in fig. 6.14 represents the mode for which the transmit interval and transmit power values are being reported.

Fig. 6.15 shows the tag reporting its sensor values. The tag carries an on board temperature sensor while it is also interfaced with an external accelerometer module. It also has the capability to attach one more analog sensor to itself. In this project no additional analog sensors have been integrated with the tag and therefore the value of analog sensor is not reported in fig. 6.15.

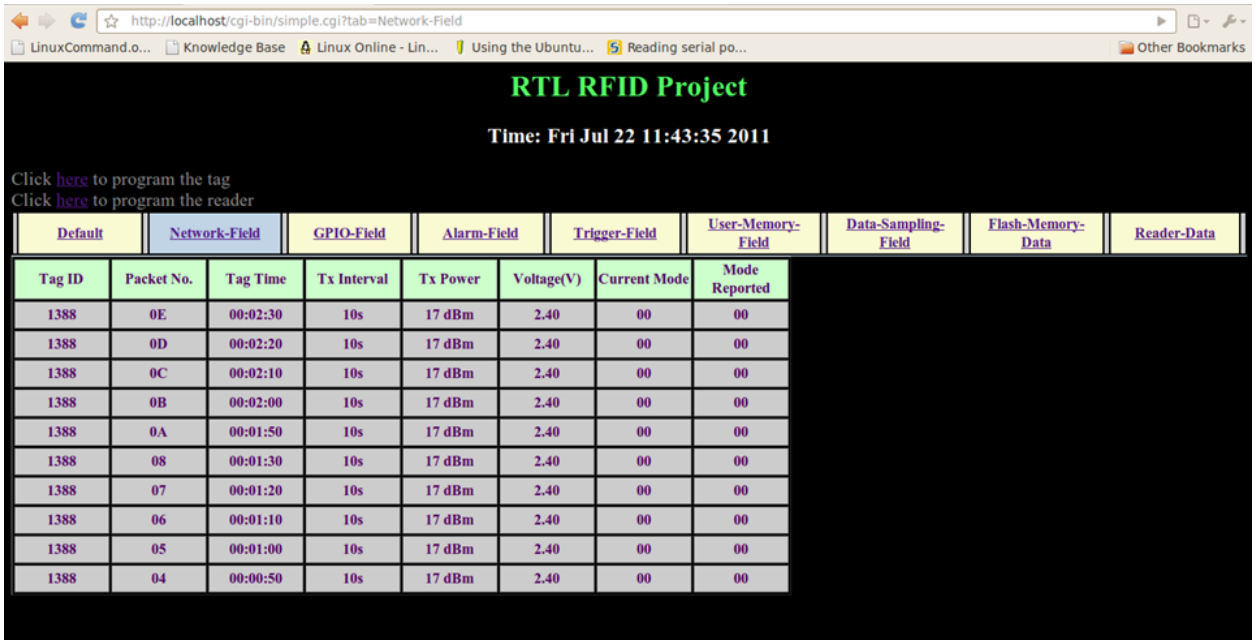


Figure 6.14: GUI showing the tag reporting its Network Field parameters

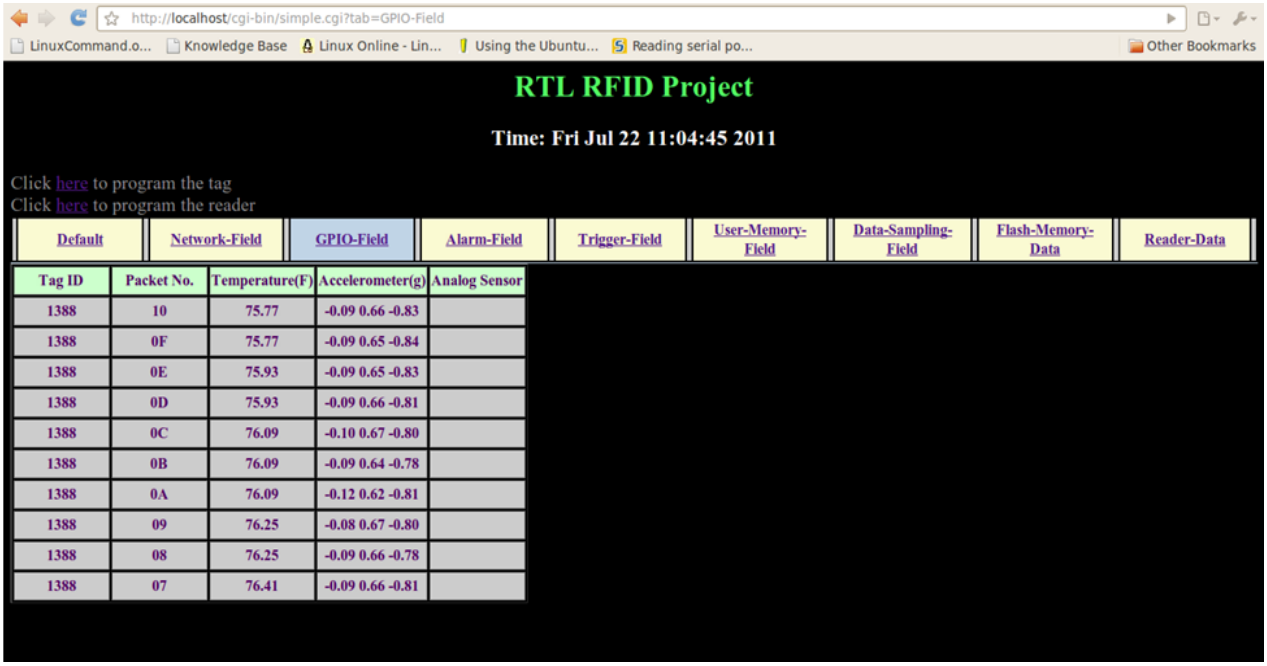


Figure 6.15: GUI showing the tag reporting its sensor values

The application layer protocol allows up to 7 programmable alarms which can be used to force the tag into a specific mode for certain duration of time. Fig. 6.16 shows how the alarm can be set. The alarm in fig. 6.16 tells the tag to switch from *Mode 0* to *Mode 7* between 12:38:00 PM and 12:38:10 PM. Fig. 6.17 shows how the *transmit interval* and *transmit power* parameters for *Mode 7* can be set. Fig. 6.18 shows the tag changing its mode and transmitting every 2 seconds between 12:38:00 PM and 12:38:10 PM in accordance with the alarm set.

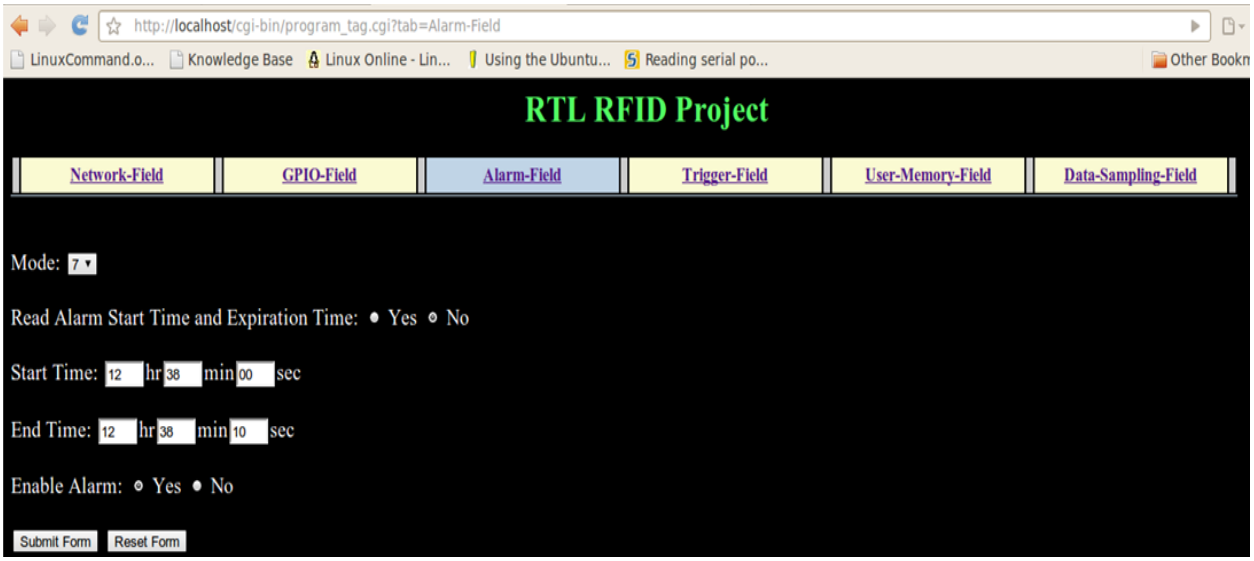


Figure 6.16: GUI used to set the alarm

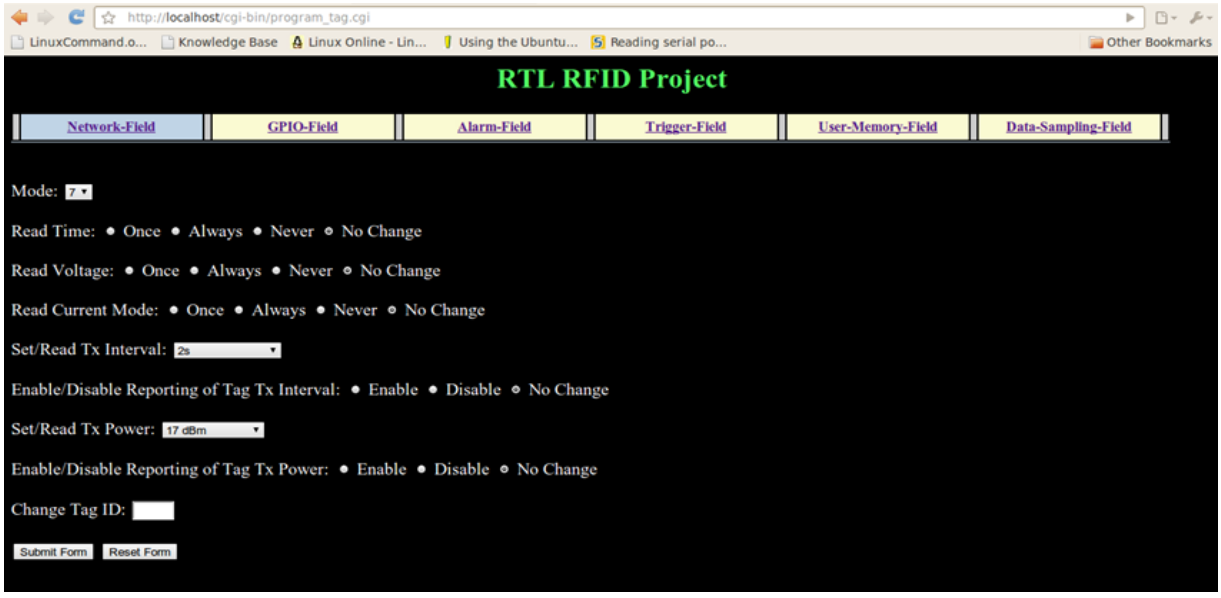


Figure 6.17: GUI used to set the transmission related parameters for a particular mode

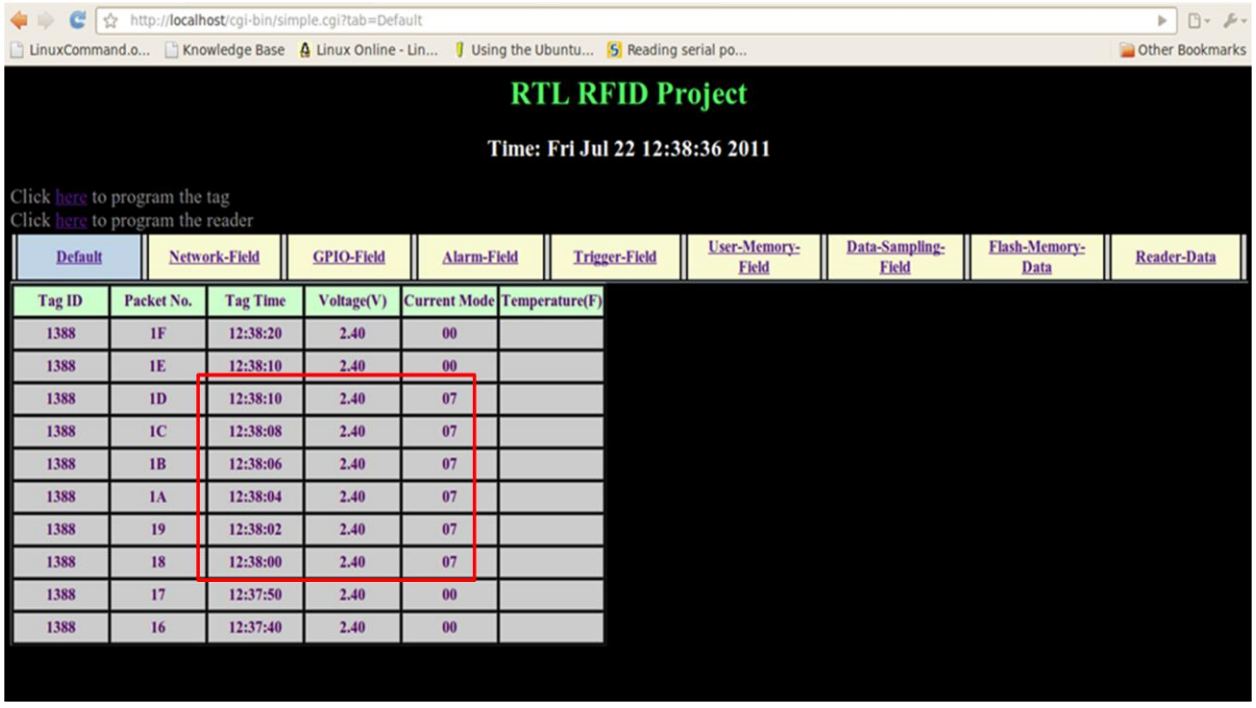


Figure 6.18: GUI showing the tag changing its mode because of the alarm

A sensor value crossing a particular threshold can also trigger the tag to change its mode. Table 6.5 shows the relationship between the temperature sensed by the temperature sensor on board the tag and the digital output from the ADC for a particular temperature range. It is clear from this table that as the temperature increases the numerical value of the output from the ADC decreases.

Decimal	Hexadecimal	Temperature (°F)
530	212	74.33
529	211	74.49
528	210	74.65
527	20F	74.81
526	20E	74.97
525	20D	75.13
524	20C	75.29
523	20B	75.45
522	20A	75.61
521	209	75.77

Table 6.5: ADC output v/s the temperature sensed by the temperature sensor on board the tag

In fig. 6.19, the lower threshold for the temperature sensor is set to be 525 which corresponds to a temperature of 75.13°F. The lower threshold target mode is selected to be *Mode 7*. Thus, the

tag is required to go to Mode 7 when the temperature goes above 75.13 ºF. Fig. 6.20 shows the tag changing its mode as per the thresholds set.

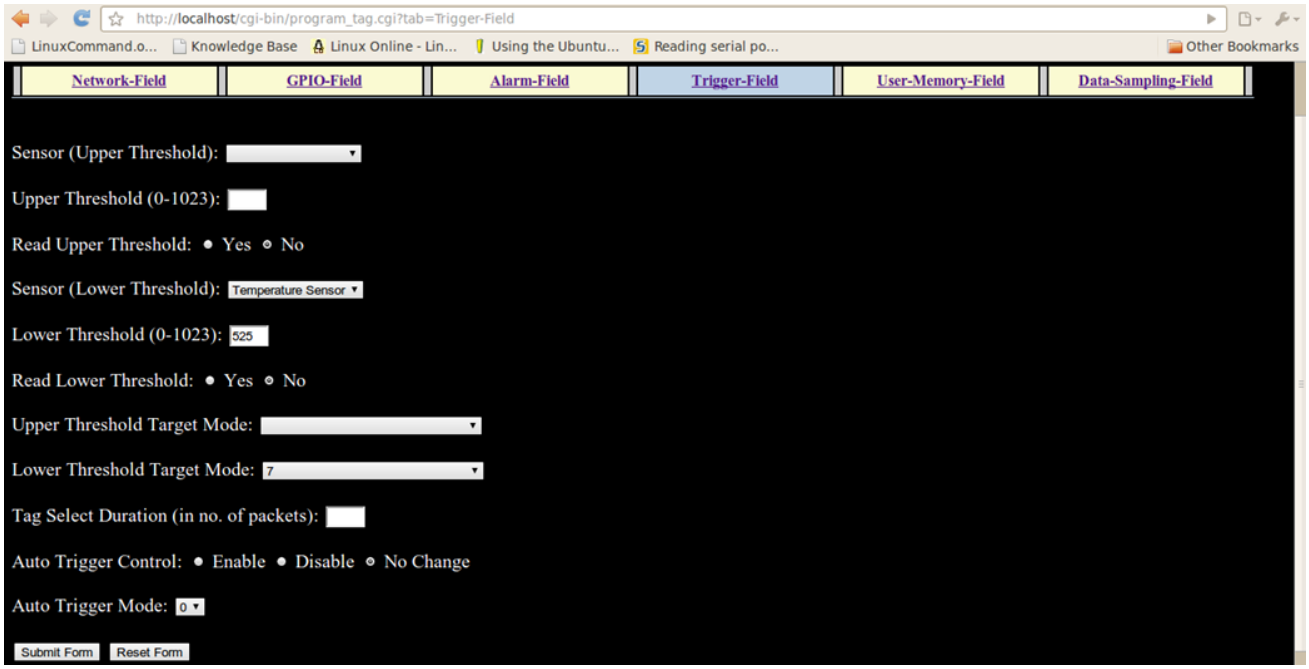


Figure 6.19: GUI used to set thresholds for triggering mode change in the tag

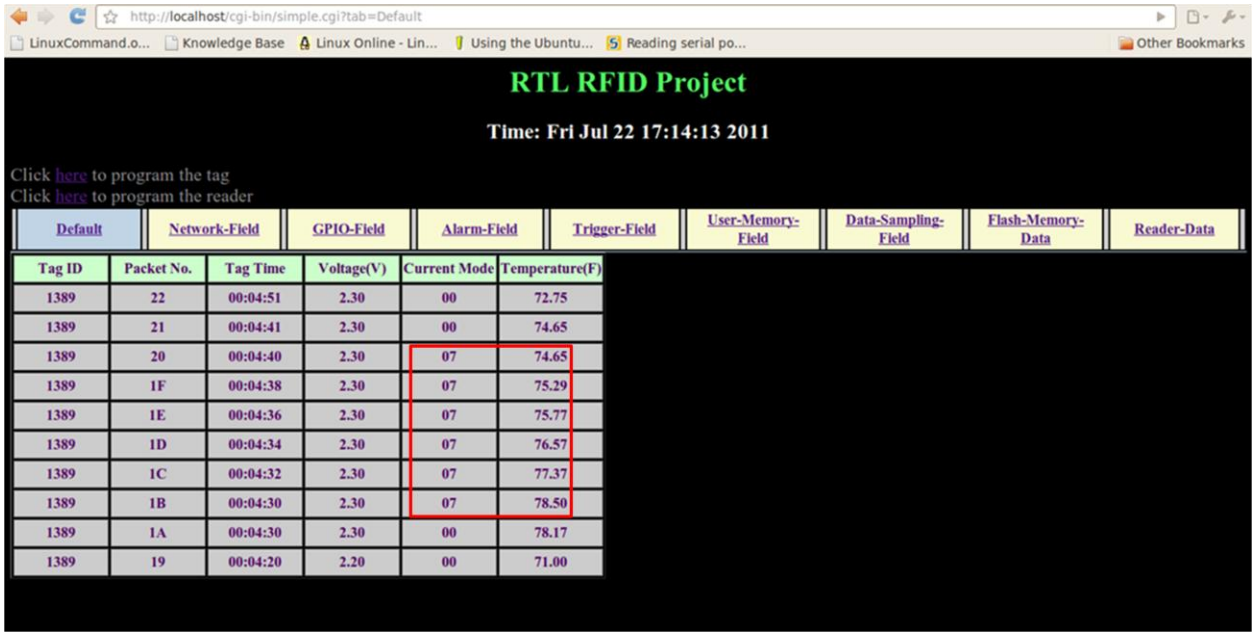


Figure 6.20: GUI showing the tag changing its mode in accordance with the thresholds set

The user can also store data in the EEPROM memory inside the tag. In fig. 6.21, 4 bytes of data are written to the EEPROM starting at byte number 0. The data written is *'AA001122'*. In fig.6.22, 3 bytes of data stored in the EEPROM starting from byte number 1 are requested by the

user. Fig. 6.23 shows the tag reporting the data stored in its EEPROM memory. The data reported back is *'001122'*.

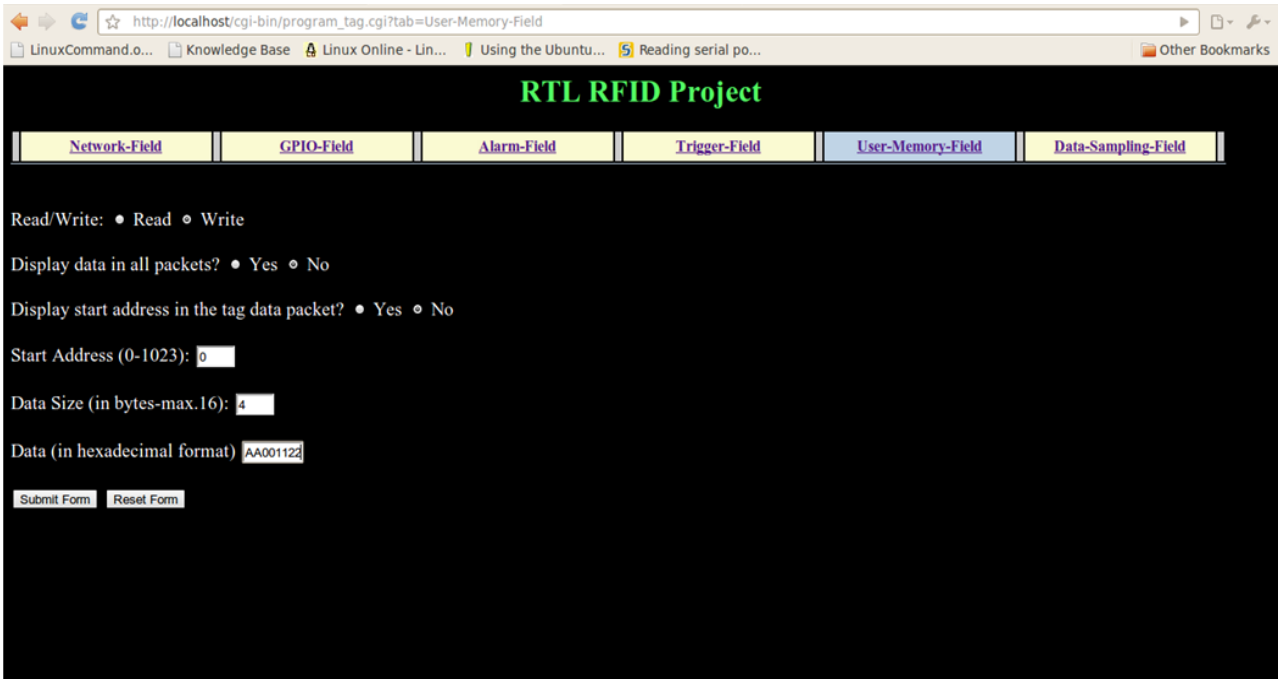


Figure 6.21: GUI to write data to the EEPROM memory inside the tag

Figure 6.22: GUI to read data from the EEPROM memory inside the tag

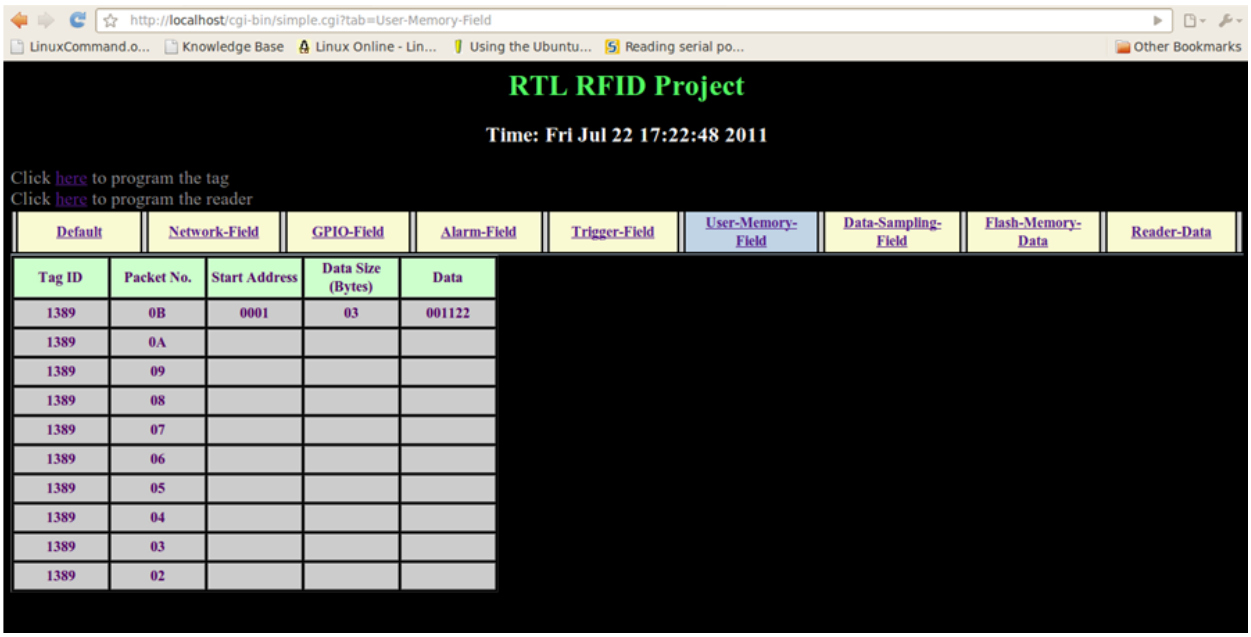


Figure 6.23: GUI showing the EEPROM data returned by the tag

The tag can also store data in its flash memory. To test this functionality, the data logging in the tag was enabled for a brief period of time and the accelerometer data was recorded. Fig. 6.24 shows the tag returning the total number of bytes stored in its flash memory after data logging has been turned off.

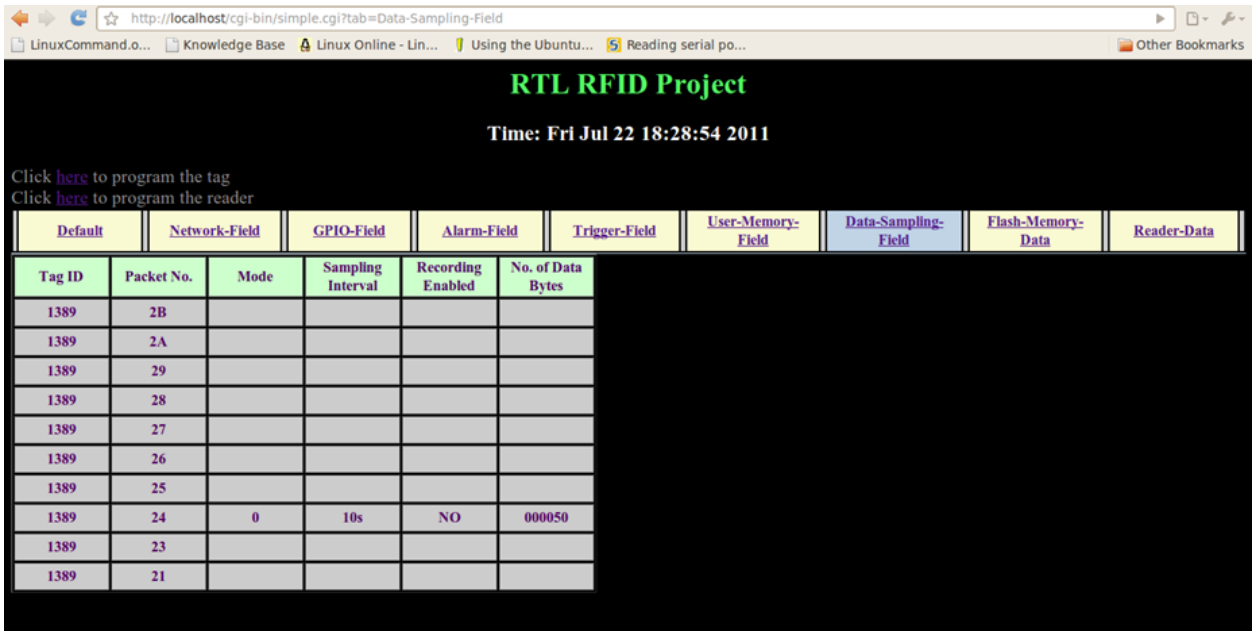


Figure 6.24: GUI showing the total number of data bytes stored inside the tag's flash memory

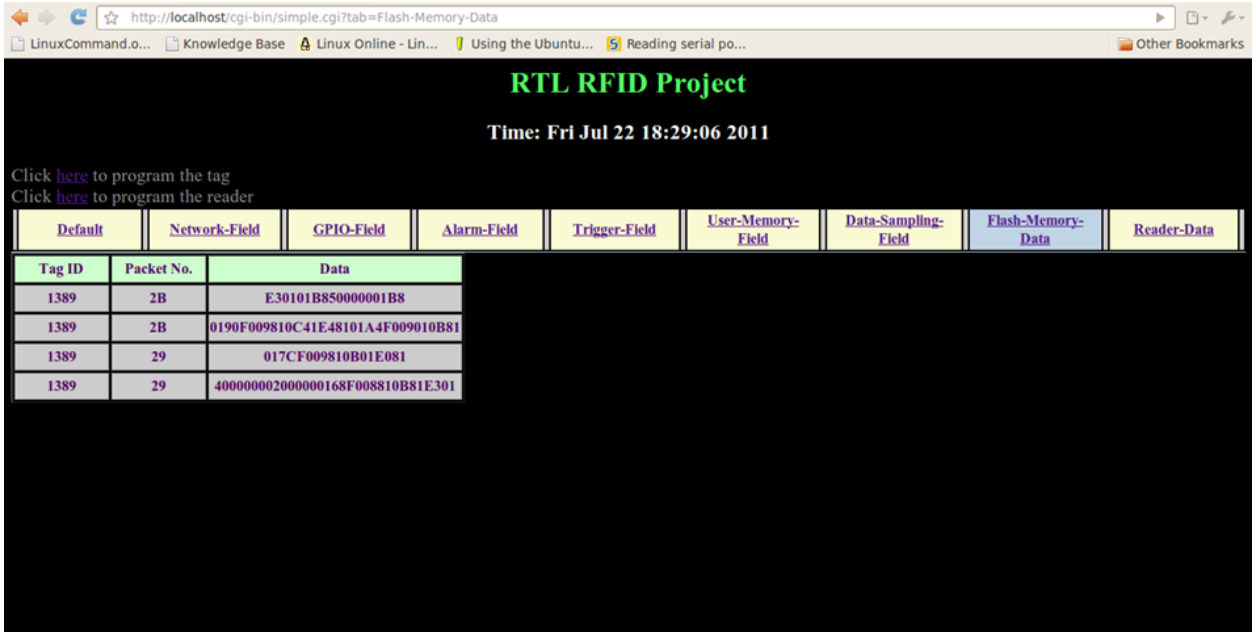


Figure 6.25: GUI showing the flash memory contents returned by the tag

Fig. 6.25 shows the flash memory data returned by the tag. The data inside the flash memory is stored in the form of data records. The data returned in fig. 6.25 contains three types of records. They are-

- *Start Recording Record:*
  400000002000000168
- *Accelerometer Data Record:*
  F008810B81E381017C
  F009810B01E0810190
  F009810C41E48101A4
  F009010B81E30101B8
- *End Recording Record:*
  50000001B8

For more details regarding these records see [18] and Appendix A.

The RFID system implemented in this project also works well in the presence of two tags. Fig.6.26 shows the reader receiving packets from two tags which can be differentiated by their tag ID.

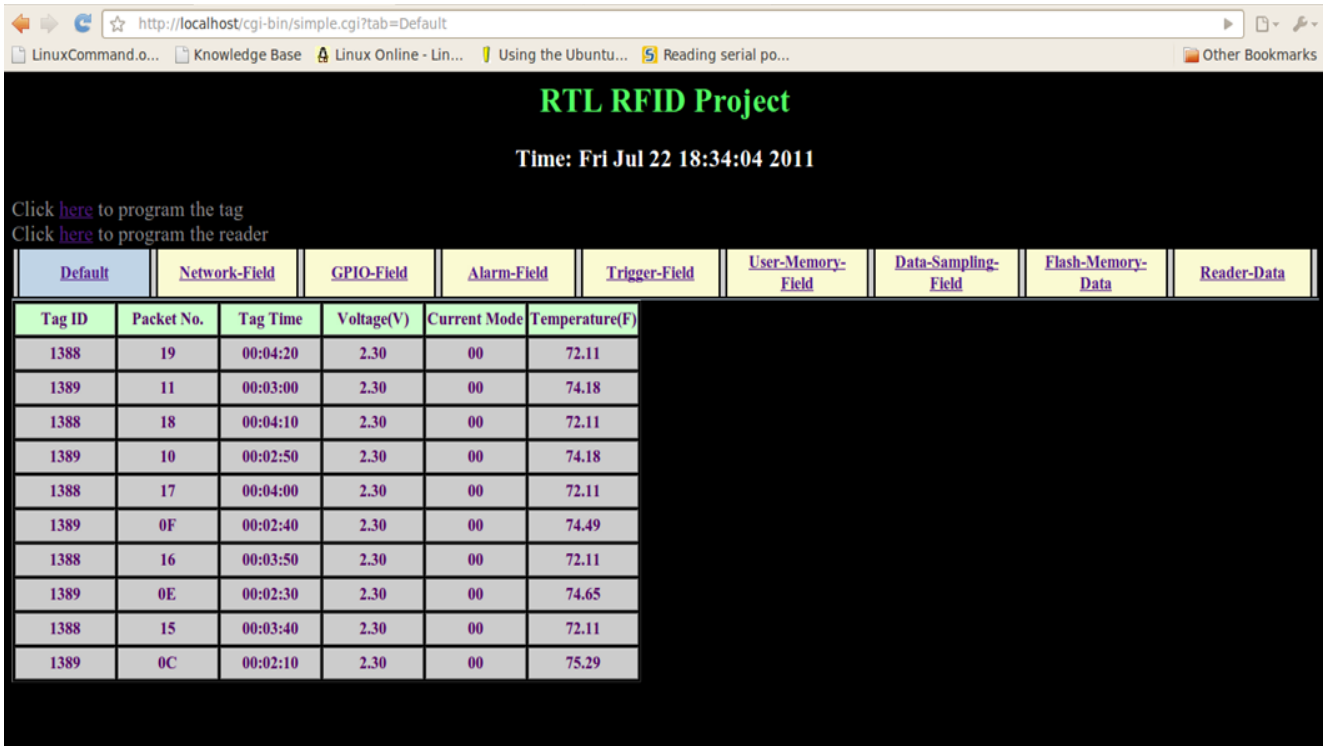


Figure 6.26: GUI showing the reception of data packets from two different tags

6.3 Testing the Accelerometer

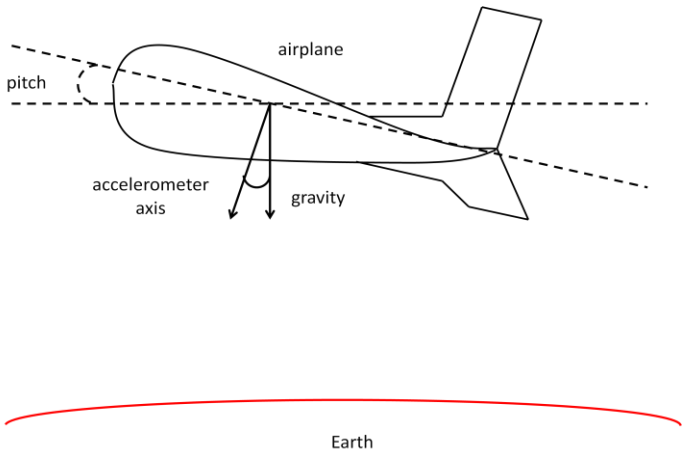


Figure 6.27: Pitch angle of an airplane [35]

The accelerometer can be used to measure the static acceleration due to earth's gravitational force. This measurement can be used to find out the pitch and roll of the device to which the accelerometer is attached. Fig. 6.27 shows the pitch angle of an airplane. The roll angle of the airplane will be the angle between the airplane and an axis perpendicular to the page. The accelerometer can also be used to measure the dynamic acceleration which can give information

regarding the way in which the object is moving [35]. Fig. 6.28 shows the axes orientation of BMA150 accelerometer.

To show that the BMA150 accelerometer was successfully interfaced with the RFID tag, the pitch and the roll of the accelerometer was measured and displayed on a GUI as shown in fig.6.29. The distance of the midpoint of the line from the center of the laptop screen (marked by a + symbol) denotes the pitch of the BMA150 accelerometer module while the tilt of the line represents the roll of the accelerator module.

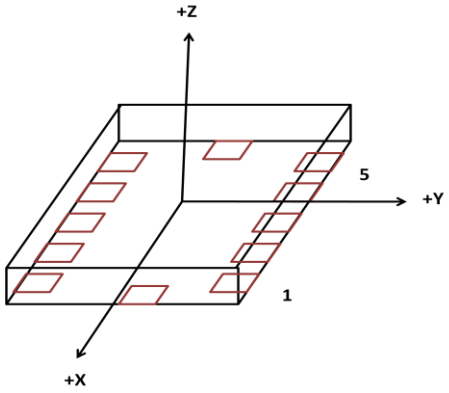


Figure 6.28: Axes orientation of BMA150 [23]

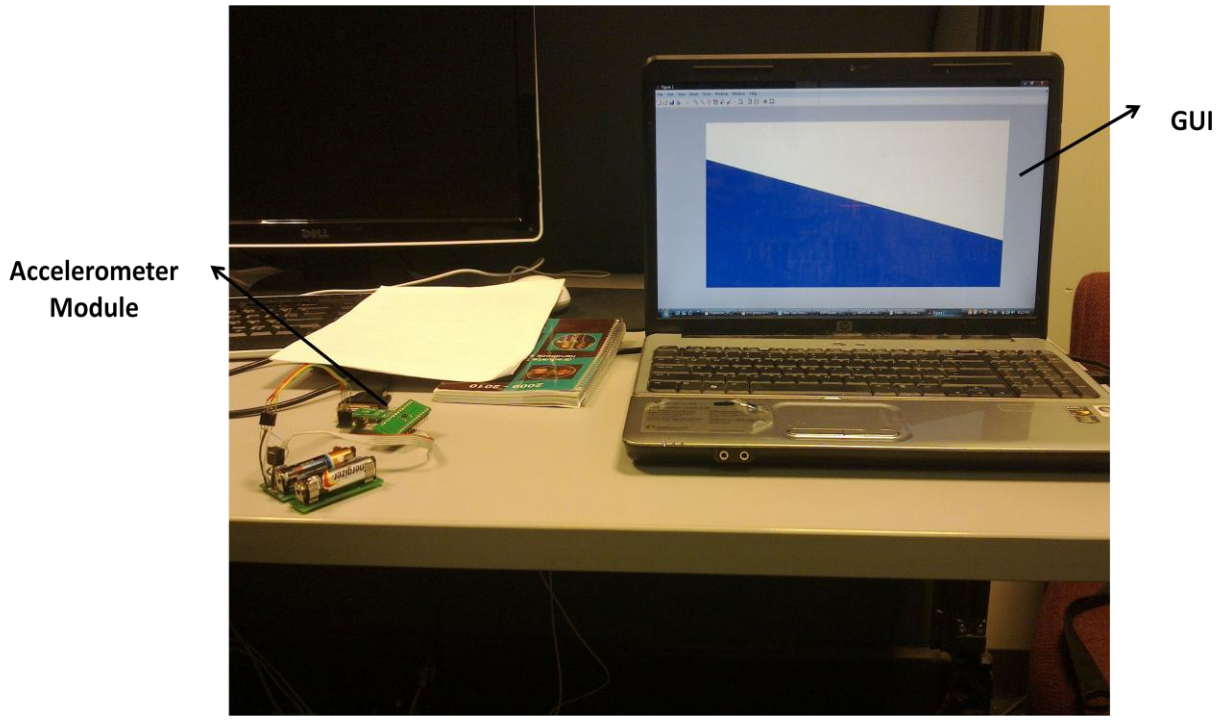


Figure 6.29: GUI showing the pitch and roll of the BMA150 accelerometer module

7 Conclusions and Future Work

This chapter presents the concluding remarks and directions for future work that can be based on this work.

7.1 Conclusion

The RFID system presented in this thesis has been shown to implement the application layer protocol from *TagSense Inc.* with some modifications. The protocol is highly configurable and is suited for various applications. The user can configure the tags over the air by sending commands to the reader. The tags have been shown to consume extremely small amounts of power in section 6.1 with the help of an oscilloscope.

The functionality of various components of the AVR ATmega328P microcontroller, the CC2520 transceiver and the CC2591 RF Front-end has been implemented with success. In addition to this, the accelerometer module has also been interfaced successfully with the tag. The implementation of a GUI with the help of CGI scripts, Apache server and Perl language has also been achieved in this thesis.

Thus the aim of this thesis to take an off the shelf tag and reprogram it to suit custom applications, in the process adding extra modules to the tag, has been accomplished.

7.2 Contributions

The major contribution of this thesis is the creation of a RFID system with all the required capabilities of the original *TagSense Inc.'s* RFID system but with extra functionalities. The system has the potential to be much more flexible as the tags and the readers can be reprogrammed. There is always a possibility of a need arising which cannot be fulfilled by the off the shelf RFID products currently available in the market. The RFID system presented in this thesis can always extend its functionality by interfacing with additional modules because of its reconfigurable nature.

The active RFID system developed in this thesis is aimed towards the needs of the railways but can easily be applied in other industries where data logging and sensor monitoring functionalities are required. This work promotes the idea of modifying products available in the market as per the needs of the application to which the system has to be applied. This allows the system to be developed in a faster manner because if a system is created from scratch then the progress will be much slower.

7.3 Future Work

As already stated in section 1.1, the real motivation for this project were the two systems required by the railways currently. Those systems have not been implemented in this thesis but the major ground work for implementing them has been laid. After implementing those two systems, they should be tested in an environment which resembles the one where the tags would actually be deployed. Right now the system has only been tested in the laboratory where the system was developed.

The range of the tags has also not been tested. Theoretically, the RFID system can provide a range up to 1700 meters when line of sight is available between the tag and the reader, but this needs to be examined.

The system has been tested to work well in the presence of only two tags. The effect of more tags on the system still needs to be investigated. As the number of tags in the system increases, the interference in the system will also increase which can hamper the communication between the tag and the reader. This can also increase the battery consumption of the tag because the tag may need to transmit multiple times before its data is received successfully by the reader.

When the data is transmitted from the tag to the reader, it is susceptible to be heard by a device which has hardware similar to the one used in this thesis. This can be avoided by encrypting the data exchanged between the tag and the reader. The CC2520 transceiver has the capability to authenticate and encrypt the communication between the tag and the reader. This capability can be used to make all the communications between the RFID devices more secure.

The RFID devices purchased from *TagSense Inc*. consist of three main components-a microcontroller, a RF transceiver and a RF front-end. Therefore, it should not be very hard to design the hardware for the RFID devices. The application layer protocol for the RFID system has already been implemented. Thus, in future it is possible to manufacture these RFID devices instead of buying them from a company.

Bibliography

1. Lee, L.T. and K.F. Tsang. *An active RFID system for railway vehicle identification and positioning*. in *Railway Engineering - Challenges for Railway Transportation in Information Age, 2008. ICRE 2008. International Conference on*. 2008.
2. (January 2002) *Active and Passive RFID:Two Distinct, But Complementary, Technologies for Real-Time Supply Chain Visibility by Savi Technology*.
3. Weinstein, R., *RFID: a technical overview and its application to the enterprise.* IT Professional, 2005. **7**(3): p. 27-33.
4. Jones, N.T., *RFID and the Difference Between Passive and Active RFID Tags.*
5. Randall, G. *Passive RFID Tags Vs. Active RFID Tags*.
6. *Active Tag (Active RFID Tag)*. Available from: www.technovelgy.com.
7. Williams, H. *Fact Sheet No. 4 - Radio Frequency Identification (Active RFID) : AIDC Centre For Wales*.
8. Harrop, P. (February 2006) *Active RFID: Innovation and Very Rapid Growth. IDTechEx Report*.
9. Amanna, A., Agrawal, A. and Manteghi, M., *Active RFID For Enhanced Railway Operations*, in *ASME 2010 Rail Transport Division Fall Conference*. October 2010: Roanoke, Virginia, USA.
10. *Wireless Sensor Network*. Available from: www.wikipedia.org.
11. Peng, Y., Q. Luo, and X. Peng. *The design of low-power wireless sensor node*. in *Instrumentation and Measurement Technology Conference (I2MTC), 2010 IEEE*. 2010.
12. Bilstrup, U. and P.A. Wiberg. *An architecture comparison between a wireless sensor network and an active RFID system*. in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. 2004.
13. Liu, H., Bolic, M., Nayak,A. and Stojmenovi,I., *Integration of RFID and Wireless Sensor Networks*. August 2008.
14. Ellingson, S., *Virginia Tech:ECE 5654-Lecture Slide.* 2010.
15. *Texas Instuments, CC2520 Datasheet*.
16. *Seem C. and Slette E., Application Note AN065: Using CC2591 Front End with CC2520*. 2009; Available from: http://focus.ti.com/lit/an/swra229a/swra229a.pdf.
17. *ZT-500 Datasheet Short Form*. 2010; Available from: www.tagsense.com.
18. *TagSense Inc., TagSense ZT-x API v3.29 Tag Functions (DRAFT)*. 2010; Available from: www.tagsense.com.
19. *TagSense Inc.,TagSense ZR-x API v3.8 Reader Functions*. 2009; Available from: www.tagsense.com.
20. Atmel, *ATmega328P AVR 8-Bit Datasheet.*
21. *Texas Instruments, CC2591 Datasheet.*
22. *AA Battery*. Available from: www.wikipedia.org.
23. *Bosch Sensortec, BMA150 Digital, triaxial acceleration sensor datasheet.*
24. *AVR Studio 4*. Available from: www.atmel.com.
25. Weddington, E.B. *WinAVR User Manual - 20050214*.
26. *<avr/io.h> AVR device-specific IO definitions*. Available from: www.nongnu.org/avr-libc/user-manual/group__avr__io.html.

27. *<avr/interrupt.h> Interrupts*. Available from: www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group__avr__interrupts.html.
28. *<avr/sleep.h> Power Management and Sleep Modes*. Available from: www.nongnu.org/avr-libc/user-manual/group__avr__sleep.html.
29. Camera, D. *Using the EEPROM memory in AVR-GCC* 2006; Available from: http://www.avrfreaks.net/.
30. *<avr/boot.h>: Bootloader Support Utilities*. Available from: http://www.nongnu.org/avr-libc/user-manual/group__avr__boot.html#ga8a60eb0985d40ff71c42bb18f0f5789e.
31. *IEEE std. 802.15.4 - 2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*. Available from: http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf.
32. Lee, J. and B. Ware, *Open Source Web Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP*. 2002.
33. *Apache Tutorial: .htaccess files*. Available from: http://httpd.apache.org/docs/2.2/howto/htaccess.html.
34. *Apache Module mod_authz_host*. Available from: http://httpd.apache.org/docs/2.2/mod/mod_authz_host.html.
35. Pycke, T. *Accelerometer to pitch and roll*. Available from: http://tom.pycke.be/mav/69/accelerometer-to-attitude.

Appendix A: Modifications Made to the TagSense Application Layer Protocol

Tag Functions

There are several changes made to the *TagSense* application layer protocol. There are some additions to the protocol while several features of the protocol have not been implemented. The real time clock onboard the tag only keeps track of the time and is reset every 24 hours. It does not keep track of the date. The acknowledgement packet transmitted by the tag back to the reader will have no return values. Also, the 2 byte additional CRC added to the command packet transmitted to the tag by the reader in the original protocol is not added in the modified version.

Network Field

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Current Time Mask Bit	Voltage Mask Bit	TX Interval Setting Mask Bit	TX Interval Report Mask Bit	TX Power Setting Mask Bit	TX Power Reporting Mask Bit	Tag ID Mask Bit 0	Current Mode Mask Bit

Table A.1: Modified TagSense Protocol-Network Field Mask Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Current Time Action Bit	Voltage Action Bit	Set/Read TX Interval Bit	Enable/Disable TX Interval Report	Set/Read TX Power Bit	Enable/Disable TX Power Report Bit	Set/Read Tag ID Bit	Current Mode Action Bit

Table A.2: Modified TagSense Protocol-Network Field Action Byte

Action Bit	Mask Bit	Meaning
0	0	No Change
0	1	Report Once
1	0	Report in all packets
1	1	Don't Report

Table A.3: Meaning of Mask and Action bits for bit 7, bit 6 and bit 0 in the modified Network Field Mask Byte and Action Byte

The modified *Network Field Mask Byte* and *Network Field Action Byte* are shown in table A.1 and table A.2 respectively. There are modifications made to the bit 7, bit 6 and bit 0. The meaning of the mask and action bits for bit 7, bit 6 and bit 0 is shown in table A.3. Also, there were changes made to the transmit power settings. The transmit power can be set by using bit 3 of *Network Field Mask Byte* and *Network Field Action Byte*. The new transmit power settings are shown in table A.4. There were no changes made to the *Network Field* for beacon packet as shown in table A.5.

Value	Tx Power [dBm]
0xF9	17
0xF0	16
0xA0	14
0x2C	11
0x03	-1
0x01	-8

Table A.4: Tx Power settings in the modified TagSense Application Layer Protocol

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Current Time Report Bit	Tx Interval Report Bit	Tx Power Report Bit	Battery Voltage Report Bit	Current Mode Report Bit	RFU	RFU	Network Field Extension Mask Bit

Table A.5: Modified Network Field for Beacon Packet

GPIO Field Byte

The Frequency Count function has not been implemented. There are two analog pins and two digital pins. The *GPIO A/D Configuration Byte* is not used but it should still be transmitted when sending a *GPIO Field* command. The modified *GPIO Mask Byte* is shown in table A.6. *GPIO I/O Configuration Byte* and *GPIO Parameter Configuration Byte* both have the same structure as shown in table A.7. Two bits are used for each sensor. The meaning of these two bits is shown in table A.8.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Temperature Sensor Pin (Pin 22 on µC)	Analog Sensor Pin (Pin 25 on µC)	Digital Pin 1 (Pin 9 on µC)	Digital Pin 2 (Pin 32 on µC)	Not Used	Not Used	Not Used	Not Used

Table A.6: Modified GPIO Mask Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Temperature Sensor Bit 1	Analog Sensor Bit 1	Digital Pin 1 Bit 1	Digital Pin2 Bit 2	Temperature Sensor Bit 0	Analog Sensor Bit 0	Digital Pin 1 Bit 0	Digital Pin2 Bit 0

Table A.7: Modified GPIO I/O Configuration Byte / GPIO Parameter Configuration Byte

Bit 1	Bit 0	Meaning	
		I/O Configuration Byte	Parameter Configuration Byte
0	0	No Change	No Change
0	1	Pin configured to be input	V_{REF}=1.1V/Output=0
1	0	No Change	No Change
1	1	Pin configured to be output	V_{REF}=3V/Output=1

Table A.8: Meaning of bit 1 and bit 0 in the modified GPIO I/O Configuration Byte and GPIO Parameter Configuration Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Temperature Sensor Bit 1	Analog Sensor Bit 1	Accelerometer Bit 1	Not Used	Temperature Sensor Bit 0	Analog Sensor Bit 0	Accelerometer Bit 0	Not Used

Table A.9: Modified GPIO Reporting Byte / GPIO Data Sampling Byte

Bit 1	Bit 0	Meaning	
		Reporting Byte	Data Sampling Byte
0	0	No Change	No Change
0	1	Report Once	Stop Recording
1	0	Report in all packets	No Change
1	1	Don't Report	Start Recording

Table A.10: Meaning of bit 1 and bit 0 in the modified GPIO Reporting Byte and GPIO Data Sampling Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Temperature Sensor	Analog Sensor	Accelerometer	Not Used	Not Used	Not Used	Not Used	Not Used

Table A.11: Modified GPIO Analog Data Report Byte for Beacon Packet

The modified *GPIO Reporting Byte* and *GPIO Data Sampling Byte* also have the same structure as shown in table A.9. Again two bits are used for each sensor. The meaning of these two bits is shown in table A.10. In the *GPIO Field* for beacon packet only *Analog Data Report Byte* is used. The modified *Analog Data Report Byte* is shown in table A.11.

Alarm Field

There are 8 alarms that can be programmed, one each for *Mode 0 – Mode 7*. The only change to the original alarm field format is that bit 3 of *Alarm Field Control Byte 1* should always be zero. Also, instead of specifying the duration of alarm in the *Alarm Field*, the expiration time of the alarm is specified.

Trigger Field

This field is also left mostly unchanged. When specifying the sensor for which the upper and lower thresholds are stated, bit 7 should be set to indicate temperature sensor and bit 6 should be set to indicate analog sensor. The *Trigger Field* for beacon packet is completely modified and is shown in table A.12.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Upper Threshold	Lower Threshold	Upper Threshold Mode	Lower Threshold Mode	Not Used	Not Used	Not Used	Not Used

Table A.12: Modified Trigger Filed for Beacon Packet

User Memory and Serial Data Field

The Serial Data Field Functions have not been implemented. The maximum amount of data that can be written to or read from the *User Memory* is 16 bytes.

Data Sampling Field

Bit 2 and bit 4 in the *Data Sampling Control Mask Byte* and *Data Sampling Action Byte* are not used. When bit 7 in *Data Sampling Control Mask Byte* is set and the same bit in *Data Sampling Action Byte* is clear then the tag not only returns its sampling interval, but also returns the total number of bytes stored in the data logger memory. The *group size* and *group index* parameters are different than those in the original protocol and have been explained already in section 4.3. The *Data Sampling Field* for beacon packet is shown in table A.13.

Byte 1	Bits 7-4 are all set to 1. Bit 3 indicates whether data recording is enabled for the mode represented by bits 2-0
Byte 2	Represents the sampling interval for the mode indicated by bits 2-0 of byte 1
Byte 3-4	Represents the total number of bytes in the data logger memory

Table A.13: Data Sampling Field for Beacon Packet

Data Records

The *Digital Data Record*, *Hybrid Data Record* and the *Power Reset Record* are not implemented. The *Analog Data Record* will contain an additional byte between *Record Header Byte* and *Record Data Bytes* which represents the *Log Indicator Byte* as shown in table A.14. Only the data corresponding to the bits set in the *Log Indicator Byte* will be recorded in the *Analog Data Record*.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Temperature Sensor	Analog Sensor	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used

Table A.14: Log Indicator Byte

The bits representing the mode in the *Header Byte* of all the record types will be the least significant 4 bits instead of the most significant bits. For example, the *Header Byte* for a *Mode Transition Record* is shown in table A.15.

Header Byte	0b01110mmm	The *mmm* indicates the current tag mode

Table A.15: Modified Header Byte for the Mode Transition Record

There is a new *Accelerometer Data Record* added to the protocol. It is explained in table A.16.

Header Byte	Value:0b11110mmm The *mmm* indicates the current tag mode.
Data Bytes	These bytes represent accelerometer data. The MSB is saved first and then the LSB. The data for X axis is stored first, then the Y axis acceleration data and at last the Z axis acceleration data is stored. There will be 6 bytes in total.
Time Bytes	Last 2 bytes of time data.

Table A.16: Accelerometer Data Record

Reader Functions

Only the following reader functions are implemented-

1. Return the number of commands in the queue.
2. Delete a command.
3. Return the command associated with a specific command handle.
4. Flush the command queue.
5. Set reader time.
6. Return reader time.
7. Append reader time to end of every tag packet that is received and forwarded to the host.