Islamic University of Technology

Ambient Intelligence: A Solution to Power Crisis

Supervised by

Md. Ashraful Alam Khan

Lecturer,

Dept. of Computer Science and Engineering,

Islamic University of Technology

Submitted by

Shafi Al Kader, ID: 124430

Eftekhar Hossain Apu, ID: 124429

List of Contents

List of Figures

Abstract

The main purpose of Ambient intelligence is to retrieve and use information from the surrounding environment to ensure optimized usage of electronic devices.

In this project we have built distributed a wireless sensor network (WSN) to detect human presence, movement and environmental changes to control electronic devices automatically in order to prevent overuse of electronic devices and decrease wastage of electricity.

The WSN was built with TelosB, a mote created by Memsic technology. It is composed of MSP430 microcontroller and CC2420 radio chip. We used TinyOS as the operating system to operate the motes. The programming language used in TinyOS is called NesC, a dialect of C.

Chapter 1: Introduction

1.1 Motivation

At present electricity crisis is one of the most severe problems of our country, because the energy infrastructure of Bangladesh is quite small, insufficient and poorly managed. Only 62% of our total population has access to electricity. Yet it is not possible to provide electricity to all of them simultaneously, because the demand is much greater than the supply. That's why Bangladesh is still reeling under 600 - 1200 MW of 'load-shedding'. A situation which deteriorates during irrigation seasons, when the demand-supply gap reaches up to 1500 MW. Every year we lose almost $1 billion and 0.5% of our national GDP due to load-shedding. [1]

Now, if we think what the causes are of this electricity crisis. We will find that there are three reasons behind it.

- Poor and backdated electricity distribution system.
- Illegal connections.
- Wastage of electricity.

The first two problems can only be solved by proper initiatives from the government. So we decided to focus only on the third problem.

Residential and industrial sectors consume about 43% and 44% electrical energy respectively, i.e. a total of about 87% of power consumption occurs in these two sectors [2]. All power sector experts acknowledge that a large part of electrical energy is consumed for lighting. Lighting and other electronic device management system in commercial buildings is very inefficient. As a result electronic devices are often turned on even though it is not necessary. This causes wastage of large amount of electricity. Besides in residential buildings people sometimes waste electricity due to lack of awareness.

These are the main reasons why a significant amount of electricity is being wasted. A way to manage the load is the introduction of "Smart and Intelligent" energy efficient electricity management system.

1.2 Impact of Efficient Energy Usage

From various sources we gathered enough data to be ensure that if an efficient energy usage system can be ensured then it can have huge impact on the electricity crisis that the world is facing right now. Some statistics below will make the impact more clear.

- 1-4% more investment in energy efficiency can save $60 billion.[3]
- Intelligent efficiency measures applied to just 35% of eligible commercial floor area in buildings can save 50 TWh by 2030.[4]
- Over 3 billion units of electricity, or a day's national consumption, were wasted in 2014-15 in India.[5]

1.3 Problem Statement

As established before, a huge amount of electricity is being wasted due to lack of awareness and inefficient management of electronic devices. These problems occur because the electronic devices are manually controlled. If the electronic devices were controlled automatically by a central intelligent management software, then the usage of electricity would be more efficient and as a result the wastage of electricity would decrease. That is why our target is to minimize the loss of electricity in our country due to poor management and lack of awareness by introducing an intelligent and efficient electronics device management system.

1.4 Proposed Solution

We are proposing an Ambient Intelligence System for efficient management of electronic device based on distributed wireless sensor network. This system will observe the state of the environment and depending on that observation it will decide which device should be turned on and which device should be turned off.

1.5 Objectives

- Building a distributed wireless sensor network for data collection.
- Developing a management software that can gather data from the WSN and process them to make intelligent decisions.
- Creating an interface in order to give the users the control to set the parameters and thresholds for data processing.

Chapter 2: Technologies We Used

We used TelosB mote to build the WSN and TinyOS as the operating system. The programming language we used is NesC.

2.1 Telosb

TelosB Mote is developed by Memsic technology. It is an open-source platform designed to enable cutting-edge experimentation for the research community. This device bundles all the essentials for lab studies into a single platform including: USB programming capability, an IEEE

802.15.4 radio with integrated antenna, a low-power MCU with extended memory and an optional sensor suite. It offers many features, including:

- IEEE 802.15.4 compliant RF transceiver
- 2.4 to 2.4835 GHz, a globally compatible ISM band
- 250 kbps data rate
- Integrated onboard antenna
- 8 MHz TI MSP430 microcontroller with 10kB RAM
- Low current consumption
- 1MB external flash for data logging
- Programming and data collection via USB
- Sensor suite including integrated light, temperature and humidity se

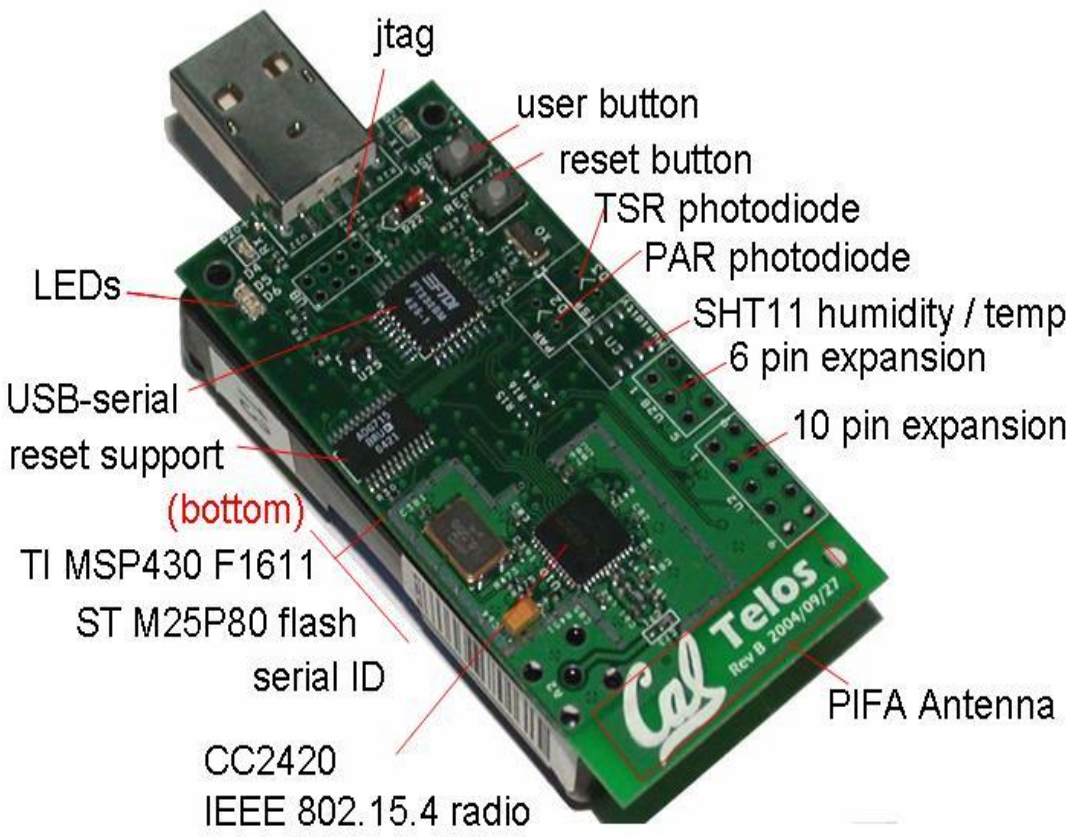

Figure: 2.1

The TelosB platform was developed and published to the research community by UC Berkeley. This platform delivers low power consumption allowing for long battery life as well as fast wakeup from sleep state. It is powered by two AA batteries. If the mote is plugged into the USB port for programming or communication, power is provided from the host computer. If the mote is always attached to the USB port no battery pack is needed. It provides users with the

capability to interface with additional devices. The two expansion connectors and onboard jumpers may be configured to control analog sensors, digital peripherals and LCD displays.

2.2 TinyOS

TinyOS is an embedded, component-based operating system and platform for low-power wireless devices, such as those used in wireless sensor networks (WSNs), smartdust, ubiquitous computing, personal area networks, building automation, and smart meters. Fundamentally, it is a work scheduler and a collection of drivers for microcontrollers and other ICs commonly used in wireless embedded platforms.

It is written in the programming language nesCas a set of cooperating tasks and processes. It began as a collaboration between the University of California, Berkeley, Intel Research, and Crossbow Technology, was released as free and open-source software under a BSD license, and has since grown into an international consortium, the TinyOS Alliance.

TinyOS applications are written in the programming language nesC, a dialect of the C language optimized for the memory limits of sensor networks. Its supplementary tools are mainly in the form of Java and shell script front-ends. Associated libraries and tools are mostly written in C.

TinyOS programs are built of software components, some of which present hardware abstractions. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation and storage.

TinyOS components offer three types of elements: Commands, Events and Tasks. These are basically normal C functions but they differ significantly in terms of who calls them and when they get called.

2.3 Relay Switch

A relay is an electrically operated switch. Many relays use an electromagnet to mechanically operate a switch, but other operating principles are also used, such as solid-state relays. Relays are used where it is necessary to control a circuit by a separate low-power signal, or where several circuits must be controlled by one signal. The first relays were used in long distance telegraph circuits as amplifiers: they repeated the signal coming in from one circuit and re-transmitted it on another circuit. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

We used Relay Switches as digital switches to turn electronic devices on/off. The basic mechanism of relay is very simple. It has 3 ports.

- Normally Connected (NC)

- Common (Comm)
- Normally Open (NO)

The Comm port is normally connected with the NC port but when voltage is applied the Comm port gets attracted by the NO port and as a result the theComm port no longer keeps connected with the NC port rather it remains connected with the NO port. So if the circuit is built in such a way that one end of the circuit is connected with the Comm port another en is connected with the NO port then normally the circuit will be open and the device will be turned off but when power applied the circuit will be complete and the device will be turned on.

Chapter 3: System Architecture

Our system has a 3 layer architecture. It has a Data Collection Layer, a Data Processing Layer, Electronics Layer and a User Interface Layer.

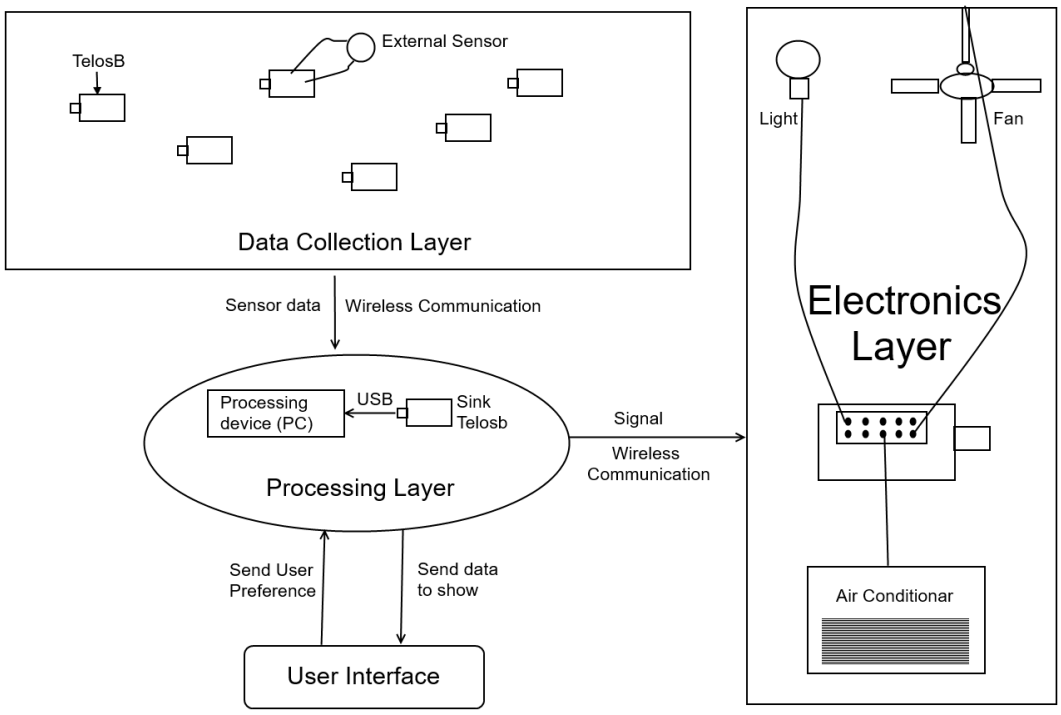

fig: System Architecture

Figure: 3.1

3.1 Data Collection Layer

This layer is for collecting data from the environment such as the temperature or the ambient light intensity. This layer consists of TelosB motes. Necessary number of motes were scattered both inside and outside the building. Each mote were powered by 2 batteries (each battery with 1.5v power). We used internal light and temperature sensor of TelosB for sensing light and temperature and we added external sensors to the expansion ports of TelosB motes for sensing other things such as human presence or movement. Collected data was directly sent to the processing layer through the internal radio of the TelosB motes without any kind of processing so that the motes don't consume extra power for processing and can last longer.

3.2 Processing Layer

This layer receives raw data from the data collection layer and processes them. This layer has a sink TelosB which can receive data from the sensor motes. This sink mote is connected to a processing device (A desktop computer in this case) through a USB port. The sink mote gets power from the computer through the USB port and also sends all the data to the computer via Serial Communication. The computer then processes all the data according to some preset parameters and threshold values and decides which electronic device should be turned on and which should be turned off.

After taking decisions the computer tells the sink mote to send on signal to the TelosB motes of the electronics layer if any electronic device needs to be turned onand to send off signal if any electronic device needs to be turned off. The sink mote sends 1 as the on signal and 0 as the off signal along with the coded name of the respective electronic devices to the TelosB motes of the electronics layers based on the decision taken by the computer.

3.3 Electronics Layer

This layer consists of all the electronic devices and some TelosB motes. The devices are connected to a digital IO pin of a TelosB mote through a relay switch. Relay switches are digitally controlled switch that breaks the circuit if it gets low voltage and completes the circuit if it gets high voltage.

A TelosB mote has several IO pins, so multiple electronic devices may be connected with one telosb. The TelosB motes of this layer receive instruction from the processing layer. The

instruction consists of an On/Off signal (1 for On, 0 means Off) and the coded name of the device that needs to be turned on/off. Each TelosB has a list of code names of electronic devices that they are connected to. They search the code name of the instruction that came from the processing layer in that list and if they get a match then they either set the IO pin that are connected to that device to high or low based on the instruction. When the relay gets the high voltage it completes the circuit and so the device is turned on and when the relay gets a low voltage it breaks the circuit and so the device is turned off.

3.4 User Interface Layer

This is the final layer of the system architecture. The main purpose of this layer is to providing the users an overview and the status of the whole system at any given time. This layer also gives the users some control to set some parameters and threshold based on which the Processing layer will take decision.

The interface is made in Java programming language. The interface contains some labels that shows the values of the reading of all the sensor. There are also some text box where the users can input a particular threshold value.

The User Interface layer communicates with the Processing layer through a process which is called Serial Forwarding. By Serial Forwarding a TelosB mote can set and get data of the java program.

Chapter 4: Methodology

4.1 Sensing

The first task of this system is to sense the changes in the environment via various sensors. Each TelosBmote has a Temperature Sensor, and a Light Sensor integrated with it. But it is also necessary to add other sensors to a TelosB mote for this system to work. We only used PIR motion sensors besides the integrated sensors. No matter which sensor we use, all the sensors can be divided in two categories:

- Analog Sensors
    - They provide the value of reading as voltage.
- Digital Sensors

- They don’t provide raw value. They only give high and low signal.

The way of sensing varies with the category of the sensors.

4.1.1 Sensing With Analog Sensors

In order to sense with an analog sensor (for example: Light Senor) we took take the following steps.

First we created a configuration file named LightSensorC.nc in the application folder.The body of the file is given below

LightSensorC.nc:

```
generic configuration LightSensorC()

{

provides interface Read<uint16_t>;

}

implementation

{

components new LightC() as LightSensor;


    Read = LightSensor;

}
```

Now in /opt/tinios2.1.2/tos/platforms/telosb we created a Configuration file name LightC.nc and added the following codes.

LightC.nc

```
generic configuration LightC() {

            provides interface Read<uint16_t>;

}

Implementation {

            components new Msp430InternalLightC();

            Read = Msp430InternalLightC.Read;      }
```

Now in /opt/tinyos-2.1.2/tos/chips/msp430/sensors we created a configuration file named Msp430InternalLightC.nc

Msp430InternalLightC.nc

```
generic configuration Msp430InternalLightC() {

provides interface Read<uint16_t>;

provides interface ReadStream<uint16_t>;

provides interface DeviceMetadata;

provides interface Resource;

provides interface ReadNow<uint16_t>;

}

implementation {

components new AdcReadClientC();

  Read = AdcReadClientC;

components new AdcReadStreamClientC();

ReadStream = AdcReadStreamClientC;

components Msp430InternalLightP;

DeviceMetadata = Msp430InternalLightP;

AdcReadClientC.AdcConfigure -> Msp430InternalLightP;

AdcReadStreamClientC.AdcConfigure -> Msp430InternalLightP;

components new AdcReadNowClientC();

  Resource = AdcReadNowClientC;

ReadNow = AdcReadNowClientC;

AdcReadNowClientC.AdcConfigure -> Msp430InternalLightP;

}
```

We also created a Module file named Msp430InternalLightP.ncIn the Msp430InternalLightP.nc add the following code.

Msp430InternalLightP.nc

```
#include "Msp430Adc12.h"

module Msp430InternalLightP {

provides interface AdcConfigure<const msp430adc12_channel_config_t*>;

provides interface DeviceMetadata;

}

implementation {

const msp430adc12_channel_config_t config = {

inch: INPUT_CHANNEL_A4,

sref: REFERENCE_VREFplus_AVss,

      ref2_5v: REFVOLT_LEVEL_1_5,

      adc12ssel: SHT_SOURCE_ACLK,

      adc12div: SHT_CLOCK_DIV_1,

sht: SAMPLE_HOLD_4_CYCLES,

sampcon_ssel: SAMPCON_SOURCE_SMCLK,

sampcon_id: SAMPCON_CLOCK_DIV_1

  };

async command const msp430adc12_channel_config_t*

AdcConfigure.getConfiguration()

  {

return&config;

  }


command uint8_t DeviceMetadata.getSignificantBits() { return 12; }

}
```

Here, the value of “inch” determines which internal sensor is being used.

For light sensor ->inch: INPUT_CHANNEL_A4

For temperature sensor ->inch: TEMPERATURE_DIODE_CHANNEL

For voltage sensor ->inch: SUPPLY_VOLTAGE_HALF_CHANNEL

For external Analog sensors we used INPUT_CHANNEL_A1 which indicates the 5th pin in the 10 pin expansion port (See fig: 4.x).

Now in the configuration file of the application add a new component like this,

SmartEnergyAppC.nc

```
newLightSensorC() as LightSensor;
```

In the module file of the application take an interface named Read<uint16_t>.

SmartEnergyC.nc

```
interface Read<uint16_t> as LightRead;
```

The Read interface provides a readDone() event which is called when reading is complete. We implemented the body of that event.

```
event void Read.readDone(error_t result, val_tval) {
        if(result == SUCCESS) {
            data = val;
        }
}
```

Now in the configuration file wire the LightRead interface to LightSensor

SmartEnergyAppC.nc

```
MVizC.LightRead ->LightSensor;
```

4.1.2 Sensing With Digital Sensors

In the configuration file of the application (SmartEnergyAppc.nc) add a component named HplMsp430GeneralIOC;

SmartEnergyAppc.nc

```
components HplMsp430GeneralIOC;

PIRTestC.IOPin -> HplMsp430GeneralIOC.Port23;
```

Port 23 is the 3rd pin in the 6 pin expansion port (See fig: 4.x).

In the module file (SmartEnergyC.nc) add an interface named HplMsp430GeneralIO.

SmartEnergyC.nc

```
uses interface HplMsp430GeneralIO as IOPin;
```

Now to get reading from Port 23 we need to set the port to input mode, and use get command.

```
callIOPin.makeInput();          //set to input mode

if(call IOPin.get())

{

        //Got high signal from sensor

}

else

{

        //Got low signal from sensor

}
```

4.2 Mote to Mote Communication

At first we created a structure to hold the data we want to send and receive.

```
typedefnx_structMoteToMoteRadioMsg {
nx_uint16_tnodeid;
nx_uint16_t reading;
} RadioMsg;
```

4.2.1 Sending

We had to take 2 components in the configuration file:

- ActiveMessageC
  and
- AMSenderC.

In the module file we used 4 interfaces:

- Packet
- AMPacket
- AMSend
- SplitControl

Also we needed a variable named pkt which is of message_t type. message_t is a structure for message buffer defined TinyOS in the Message.h file.

In the Boot.booted() method of the module file we called AMControl.start() method.

```
event void Boot.booted() {
callAMControl.start();
  }
```

The AMControl interface provides two events called startDone() and stopDone(). We had to implement the body of the event startDone(). This event is called when AMControl starts.

```
event void AMControl.startDone(error_t err) {
if (err == SUCCESS) {
call Timer0.startPeriodic(500);// We will send data in each 500
        milliseconds
   }
else {
callAMControl.start();
   }
  }

event void AMControl.stopDone(error_t err) {
  }
```

In the fire event of Timer0 we wrote the code for sending data.

```
event void Timer0.fired() {
  ...
if (!busy) {
RadioMsg* btrpkt = (RadioMsg*)(call Packet.getPayload(&pkt, sizeof
        (RadioMsg)));
btrpkt->nodeid = TOS_NODE_ID;
btrpkt->reading = data;
if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(BlinkToRadioMsg)) ==
        SUCCESS) {
busy = TRUE;
   }
  }
}
```

Here busy is a Boolean variable. Its default value is false.

The AMSend interface provides an event called sendDone() which is called after a message is sent. We implemented the body of that event.

```
event void AMSend.sendDone(message_t* msg, error_t error) {
if (&pkt == msg) {
busy = FALSE;
```

```
  }
 }
```

In the configuration file we need to wire the components with the interfaces like this

```
SmartEnergyC.Packet ->AMSenderC;
SmartEnergyC.AMPacket ->AMSenderC;
SmartEnergyC.AMSend ->AMSenderC;
SmartEnergyC.AMControl ->ActiveMessageC;
```

4.2.2 Receiving

In the configuration file we took a component called AMReceiver and in the module file we used an interface called Receive.

The Receive interface provides an event called receive which is called when a message is received. We implemented the body of that event.

```
eventmessage_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
if (len == sizeof(RadioMsg)) {
RadioMsg* btrpkt = (RadioMsg*)payload;
Data = btrpkt->counter;
  }
returnmsg;
}
```

In the configuration file we wired the components to the interfaces like this:

```
implementation {
  ...
SmartEnergyC.Receive ->AMReceiverC;
}
```

4.3 Connecting Hardwares

4.3.1 In Data Collection Layer

Data collection layer consists of some sensors and TelosB mote. The sensors are connected with TelosB. There are 3 wires in almost every sensors. They are

- VCC
- Ground
- Data

The VCC and Ground of sensors will be connected with the VCC and Ground port of TelosB. The Data of sensors will be connected with the expansion analog or digital port of TelosB.

4.3.2In Data Processing Layer

Data processing layer consists of a sink TelosB and a computer. This TelosB has a unique ID. The Sink is connected with the computer through USB port.

4.3.3 In Electronics Layer

In the electronics layer all the devices are connected with 5V relay and the relay will collect signal from a TelosB.

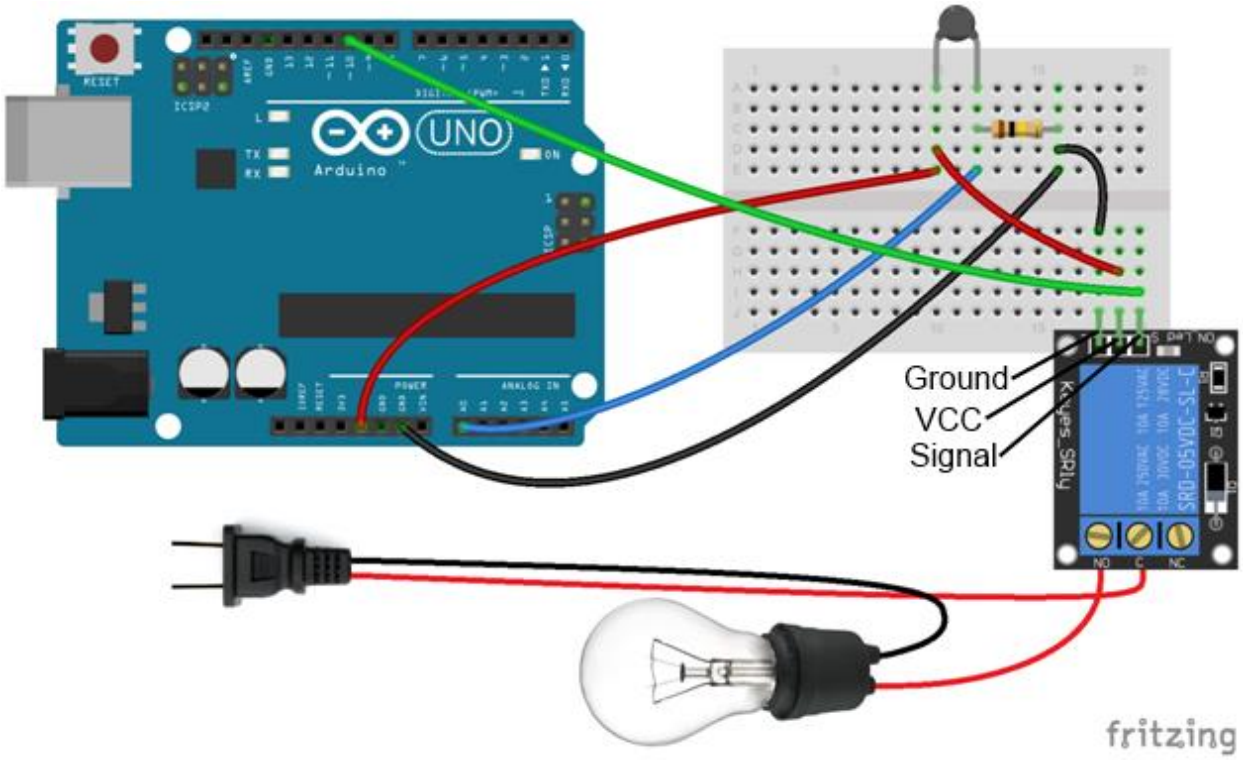

Figure: 4.1

Here the VCC port of relay will be connected with a 5V power source and the ground part should be connected with a ground. The data port of relay will be connected to an output port of TelosB. In the figure the Arduino will be replaced by a TelosB. On the outside ports of relay, the 'com' port of relay will be connected directly to power source with wire. The 'NO' port of relay will be connected to the device with a wire and another wire coming from the device will be connected with the power.

4.3.4 Circuit Diagram

The wired connections between devices are described in this section with figures and diagrams

External Digital Sensor with TelosB

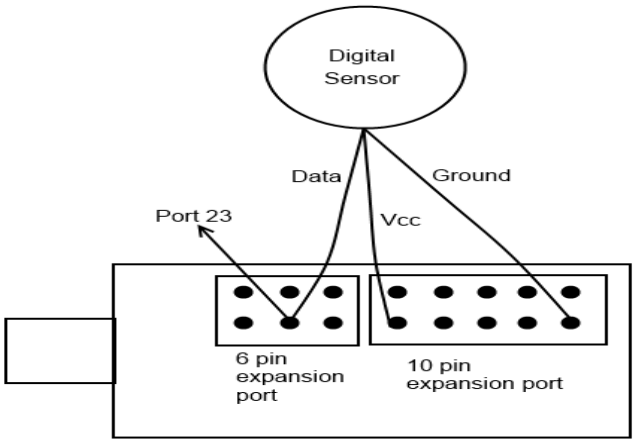

Figure: 4.2

ExternalAnalog Sensor with TelosB

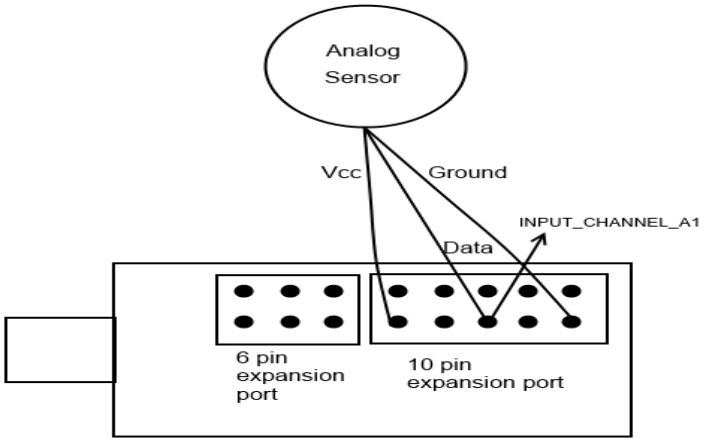

Figure: 4.3

Relay with Device and TelosB

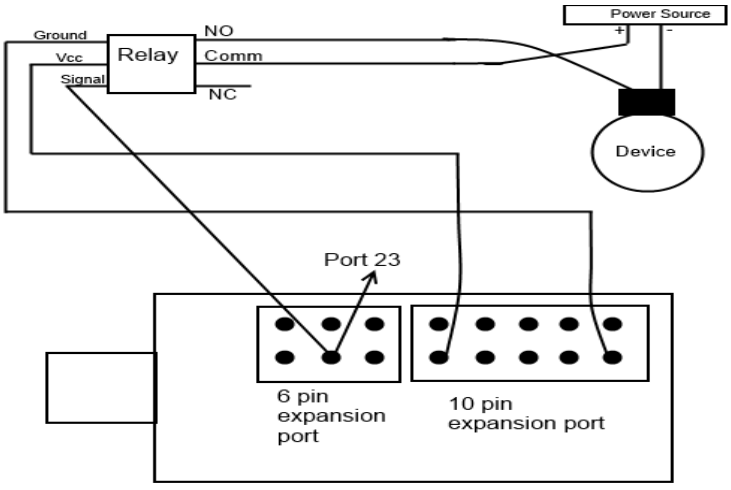

Figure: 4.4

Chapter 5: Conclusion

5.1 Summary

We built a Distributed Wireless Sensor Network to develop a Smart and Efficient Electronics Device Management System with Sensors and TelosB in TinyOS platform in order to reduce the wastage of electricity.

5.2 Advantages

- We will take our decision based on collective data.
- The decision will be Smart because it can adapt to every scenario.
- We can use customized protocol of data transmission.
- It provides secured data transmission.

5.3 Things We Did

- Reading data from analog sensors of TelosB.
- Adding external digital Sensors in TelosB.
- Analog to digital conversion.
- Establishing communication network.
- Collaboration between different sensors.
- Developing a complete distribution network system.
- Smart decision making.
- Sensor calibration for getting perfect result.
- A user friendly interface to control all the sensors and devices.

5.4 Future Implementations

- Detecting people while sleeping.
- Machine learning for devices.
- More cost efficient.
- Making the interface more user friendly.
- Making an application for smart mobile so that user can control the devices from anywhere.

References:

1. Power Crisis and Solution in Bangladesh authored by Mohammad AsadulHaque, JalalurRahman.

2. https://en.m.wikipedia.org/wiki/Electricity_sector_in_Bangladesh

3. According to a survey of American Council for an Energy Efficient Economy in 2013.

4. http://www.buildings.com/article-details/articleid/19537/title/how-smart-buildings-save-energy.aspx

5. http://timesofindia.indiatimes.com/india/Three-billion-units-of-power-wasted-in-one-year/articleshow/47942237.cms