**A Simple Wireless Sensor Network based on the Arduino and XBee**

**Dino Tinitigan**

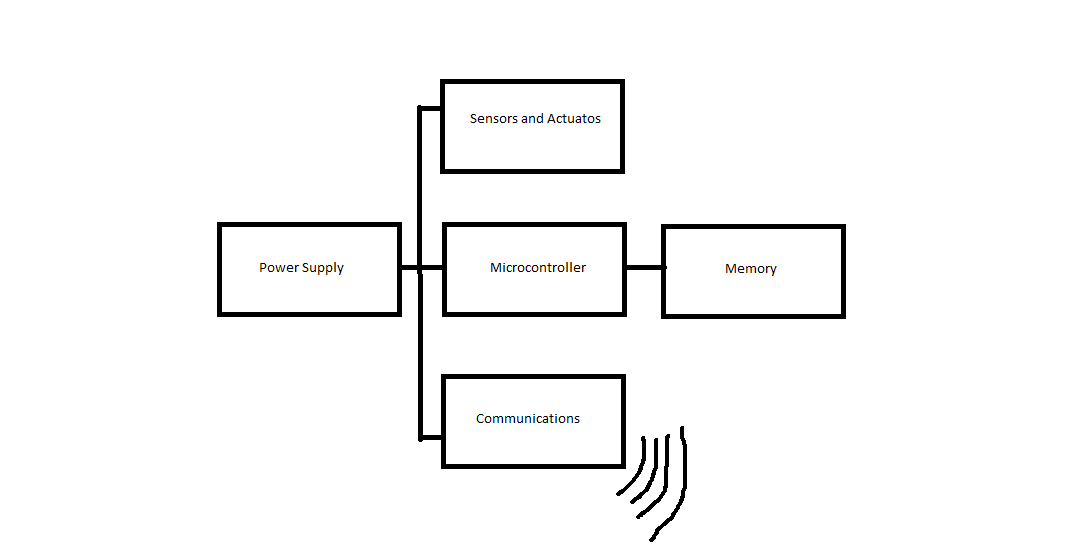
***Abstract* – Wireless Sensor Networks Overview and implementation using Arduino and XBee.**

I. Introduction

Due to the advances in different technologies, WSNs(Wireless Sensor Networks) are becoming more affordable and more practical. WSNs are can be used to monitor environmental conditions such as temperature, pressure, sound, vibration, etc. It is particularly useful for areas where there is no easy access to a tethered power source.

II. WSN Architecture

WSNs generally consist of the following: a controller, memory, sensors and actuators, communications, and a power supply [1].

  
**Fig 2.1 WSN Architecture**

**Controller –** The controller is the main devices that processes the data and executes the code. Typically a microcontroller is used because they have sleep states which enable the node to save power. Two of the most common microcontrollers are manufactured by Atmel (AVRs) or Micron (Pics).

**Memory –** A place to store data is also necessary. Program data and sensor data are usually stored separately, but not necessary. Microcontrollers have built in memory(RAM) and can be used to store data before transmission. External EEPROM(Electrically Erasable Programmable Read-Only Memory) or flash memory can also be used if RAM is not enough.

**Sensors and Actuators –** Sensors and actuators are used to interface with the environment. Sensors can detect measure parameters such as temperature level. Actuators can be used to control a mechanical system or part.

**Communications –** Responsible for sending and receiving data reliably over a wireless channel. For actual communication a transmitter and receiver is need in a node. Usually half-duplex is used because communication both ways is impractical[1].

**Power Supply –** The power supply is responsible for delivering enough power to the node. The power supply is normally a battery. Sometime energy is also harvested from the environment such as a solar cell.

III. Topologies

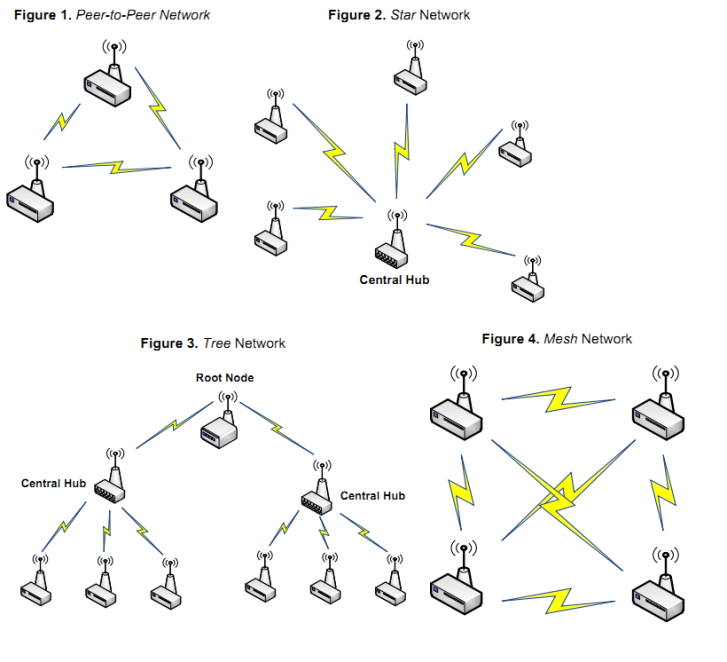
There are generally four (4) basic types of WSN topologies. They are Point to Point, Star, Tree, and Mesh[2]. There are more sophisticated topologies but they will not be discussed in this paper.

**Point to Point –** Point to Point allows node to communications of a node with another node without the need of a centralized server or coordinator.

**Star –** Star networks have a centralized hub or coordinator that each node must communicate to. Each node is unable to directly communicate with each other.

**Tree –** Tree networks can be considered as a hybrid of Star and Peer to Peer[2]. In a tree the top level is called the root. One level lower would be a central hub or coordinators which forms a Star network.

**Mesh** – In a Mesh network, nodes allow data to hop from each other node. Each node is able to communicate with each other. It also allows for self-healing networks but is also very complicated and expensive to implement.

  
**Fig 3.1 WSN Topologies**

IV. Power Management

Since nodes are generally powered by batteries and replacing them is impractical, energy consumption must be kept as low as possible.

**Sleep Modes –** In order to conserve power, the microcontroller in the node needs to be put in sleep mode. There different levels of sleep depending on the microcontroller. Typically the deepest sleep levels consume only a few uW, however some sleep mode can only be woken by external interrupts. Some transceivers also have sleep modes. For example the XBee series 1 module has a few different sleep modes depending on the application.

**Dynamic Power Scaling –** The idea of dynamic power scaling is to choose the slowest and lowest speed and voltage setting required for the application. It takes advantage of the fact that microcontrollers consume less power at lower speeds and lower operating voltages.

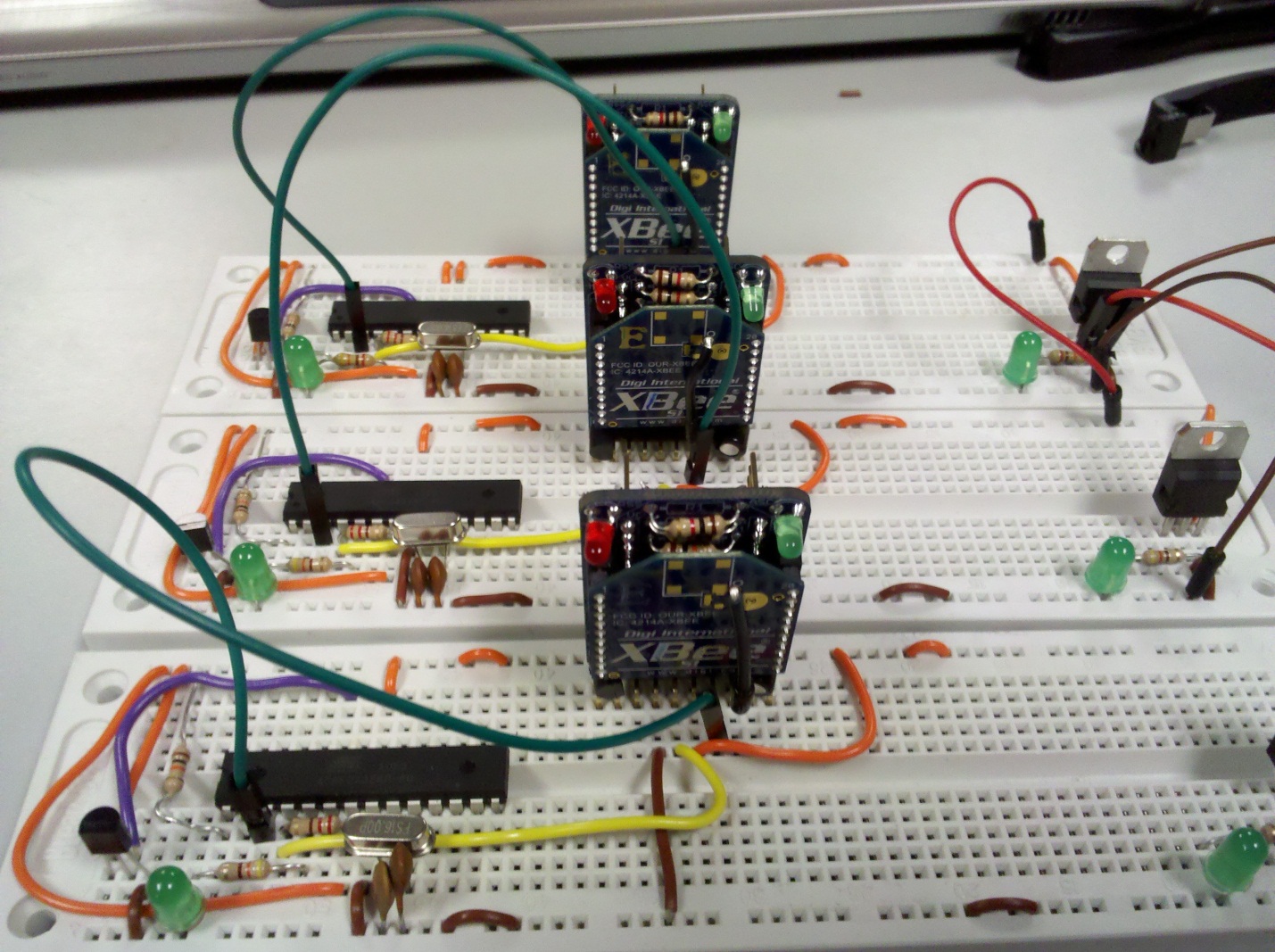
**Energy Scavenging** – The main concept of energy scavenging is to gather energy from the environment. One of the most common form of this is using photovoltaics(solar cells. Other possible sources of energy that can be used are vibrations, temperature gradients, wind, pressure variations, etc.[1].

V. Implementation

My implementation of a WSN would focus on a simple Star topology with some outer nodes, each gathering the surrounding temperature. Data will be transferred using XBee radios. The remote nodes and remote Xbee modules will also be put into sleep mode when needed to conserve power. The main objective for the implementation would be to build a simple WSN and show a proof of concept.

* **Controller** – Atmega328 microcontroller.
* **Sensors** – TMP36 temperature sensor.
* **Communications** – XBee module on each node.
* **Power Supply** – 3x AA Batteries on each node.
* **Topology** – Star
* **Power saving** – Sleep modes for microcontroller and XBee when not transmitting or receiving.

**Nodes** – Each node would contain a temperature sensor, a microcontroller, an XBee, and batteries as a power source. Each node would also have power saving features such as sleep modes.

  
**Fig 5.1 Prototype Remote Node**

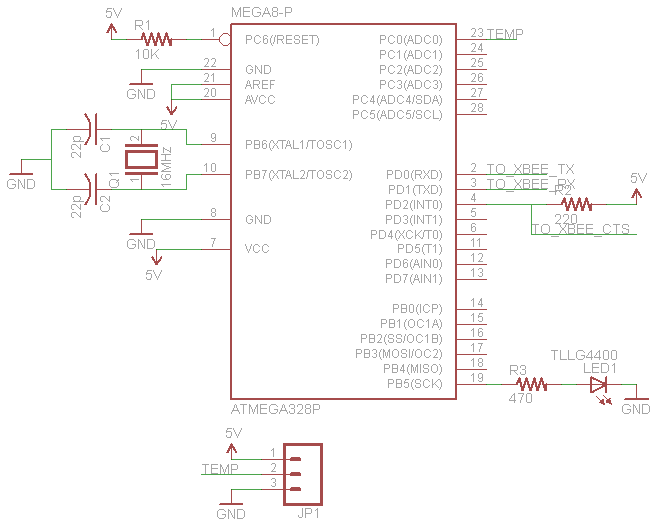
**Coordinator** – The main coordinator would simply consist of an XBee connected to a PC. The PC would have software that control data logging. The main job of the coordinator is to wait for incoming data from the remote nodes.

**PC Software** – The PC software will be responsible for showing the data visually to the user. Additionally it will feature data logging features by saving the gathered data to a file which can be used for later data analysis.

**Power Savings –** By using sleep mode on both the microcontroller and the XBee modules, I was able to achieve some great power savings. Normal power consumption without any sleep modes is 0.387 W. When both the microcontroller and the XBee module is asleep power consumption is at 0.0495 W. This translates roughly a power reduction factor of 1/8 or 12.5% of the original. Power savings can be further improved by choosing a different microcontroller and lowering the operating voltage. However, this is only a prototype and therefore, I used parts that were readily available to me. Also power consumption is greatly dependent on the frequency of transmission. If data is only needed to be transmitted once every hour, then the nodes will only need to be awake once/hour. If data is needed to be transmitted once every 5 minutes then the nodes will have to wake up every 5 minutes.

VI. Design

**Arduino** – In order to simplify the design an Arduino was used to program and test each of the outer nodes.

  
**Fig 6.1 Schematic diagram of a node with a temperature sensor**

**XBee Configuration** – In order to allow the XBee module to enter sleep modes and save power, it needs to be configured properly. The coordinator XBee is set to be awake at all time and simply just waiting for data. The remote XBee are set to be on cyclic sleep mode. This allows the Xbee to sleep for a set amount of time. When the XBee wakes up, it pulls the CTS pin low which can then be used to wake the microcontroller up.

**Custom Protocol** – In order reliably send data between the coordinator and nodes a custom packet was created. The first byte is the header byte with a fixed value of 0xFF, the next is the node ID, then it is followed by the data bytes, and the last byte is a checksum byte.



**Custom PC Program** – A custom PC software was programmed using processing. It is designed to simply listen to the serial port. Once the program sees a header byte ‘0xFF’, it knows it the beginning of a packet and it does the appropriate calculations to determine the ID and temperature of the remote nodes. It also does the checksum calculations to check if there any errors in the transmission. The software is also designed to show a simple visual interface and display the temperature reading of the individual remote nodes. Additionally the software is designed to handle data logging. It also tags the data with the real time, and time since star. Data is comma delimited to allow easy data analysis with spreadsheet program such as MS Excel.

VII. Data Logging

**Graph 7.1 Sample Data Logged**

Graph 7.1 shows a sample of data gathered over a short period of time. All the nodes were located in the same room. Artificial temperature spike were created using a soldering iron. As expected, the data is uniform among the three nodes because they are all in the same area.

VIII. Source Code

**Remote Node:**

|  |
| --- |
| //Dino Tinitigan  //Sleeping Node  //Sleep code derived from sleep code from Arduino website  //Sends temperature data to coordinator  //Goes back to sleep after sending data  #include <avr/sleep.h>  int wakePin = 2; // pin used for waking up  int sleepStatus = 0; // variable to store a request for sleep  int temp1;  int ID = 6; //ID of node, change this!!!  int ledPin = 3;  double t1;  int temp2;  double vref = 5.04;  void wakeUpNow() // here the interrupt is handled after wakeup  {  // execute code here after wake-up before returning to the loop() function  // timers and code using timers (serial.print and more...) will not work here.  // we don't really need to execute any special functions here, since we  // just want the thing to wake up  }  void setup()  {  pinMode(ledPin, OUTPUT);    pinMode(wakePin, INPUT);  Serial.begin(9600);  attachInterrupt(0, wakeUpNow, LOW); // use interrupt 0 (pin 2) and run function  // wakeUpNow when pin 2 gets LOW  }  void sleepNow() // here we put the arduino to sleep  {  //Serial.println("Sleeping");  digitalWrite(ledPin, LOW);  set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN); // sleep mode is set here  sleep\_enable(); // enables the sleep bit in the mcucr register  // so sleep is possible. just a safety pin  attachInterrupt(0,wakeUpNow, LOW); // use interrupt 0 (pin 2) and run function  // wakeUpNow when pin 2 gets LOW  sleep\_mode(); // here the device is actually put to sleep!!  // THE PROGRAM CONTINUES FROM HERE AFTER WAKING UP  sleep\_disable(); // first thing after waking from sleep:  // disable sleep...  detachInterrupt(0); // disables interrupt 0 on pin 2 so the  // wakeUpNow code will not be executed  // during normal running time.  }  void loop()  {  digitalWrite(ledPin, HIGH); //turn led on while awake  // display information about the counter  temp1 = getTemp();  byte checksum = ID + temp1;    Serial.write(255); //header  Serial.write(ID); //ID BYTE  Serial.write(temp1); //DATA BYTE/S  Serial.write(checksum);    //Serial.print("I am ");  //Serial.println(ID);  delay(3000);  sleepNow();  }  int getTemp()  {  t1 = analogRead(0);  t1 = t1\*1000; //millivolts factor  t1 = t1/1024; //10-bit range  t1 = t1\*vref; //vref  temp2 = (t1-500)/10;  return temp2;  } |

**Coordinator Node:**

|  |
| --- |
| //Dino Tinitigan  //CPE 400  //Coordinator Application  import processing.serial.\*;  int incoming;  int nodeID;  int value;  int numMembers = 5;  int[] members = {2,3,4,5,6};  int[] values = new int[5];  int checksum;  int t = 5;  int ID;  Serial myPort;  //----File IO---------  //use hour(), minute(), seconds();  //ude date()  PrintWriter output;  void setup()  {  myPort = new Serial(this, Serial.list()[0], 9600);  println(Serial.list());    textSize(30);  size(800, 600);  background(80, 120, 255);    //-------------------setup File IO-------------------  output = createWriter("log.csv");    output.print("Date and Time," + "millis since start,");  for(int i = 0; i < numMembers; i++)  {  output.print("Node " + i + ",");  }  output.println();  //-------------------setup File IO-------------------  }  void draw()  {  if(myPort.available()>0) //data is on serial port  {  incoming = myPort.read();  println("Data Received");  while((incoming != 255) && myPort.available()>0) //look for header if their is data on serial port  {  incoming = myPort.read();  }    if(incoming == 255) //header found  {  //header received  println("----------header received-----------");  nodeID = myPort.read(); //read ID  value = myPort.read(); //read data  println("Node " + nodeID);  println(value + " C");  updateValue(nodeID, value);  }  }  //-------------------------------------------------------------------------------------------------------  }  void updateValue(int ID, int val)  {  switch(ID)  {  case 2:  {  int check = myPort.read();  checksum = ID + val;  if(checksum >=256)  {  checksum = checksum - 256;  }  if(checksum == check)  {  //draw box  fill(0,255,0);  rect(10,10,200, 100);  fill(0);  text("Node: " + ID, 30, 40);  text(val + " C", 30, 100);  output.print(month() + "/" + day() + "/" + year() + "/" + hour() + ":" + minute() + ":" + second() + "," + millis());  output.println("," + val);  }  break;  }  case 3:  {  int check = myPort.read();  checksum = ID + val;  if(checksum >=256)  {  checksum = checksum - 256;  }  if(checksum == check)  {  fill(0,255,0);  rect(10,200,200, 100);  fill(0);  text("Node: " + ID, 30, 225);  text(val + " C", 30, 285);  output.print(month() + "/" + day() + "/" + year() + "/" + hour() + ":" + minute() + ":" + second() + "," + millis());  output.println(",," + val);  }  break;  }  case 4:  {  int check = myPort.read();  checksum = ID + val;  if(checksum >=256)  {  checksum = checksum - 256;  }  if(checksum == check)  {  //draw box  fill(0,255,0);  rect(200,10,200, 100);  fill(0);  text("Node: " + ID, 220, 40);  text(val + " C", 220, 100);  output.print(month() + "/" + day() + "/" + year() + "/" + hour() + ":" + minute() + ":" + second() + "," + millis());  output.println(",,," + val);  }  break;  }  case 5:  {  int check = myPort.read();  checksum = ID + val;  if(checksum >=256)  {  checksum = checksum - 256;  }  if(checksum == check)  {  fill(0,255,0);  rect(200,200,200, 100);  fill(0);  text("Node: " + ID, 220, 225);  text(val + " C", 220, 285);  output.print(month() + "/" + day() + "/" + year() + "/" + hour() + ":" + minute() + ":" + second() + "," + millis());  output.println(",,,," + val);  }  break;  }  case 6:  {  int check = myPort.read();  checksum = ID + val;  if(checksum >=256)  {  checksum = checksum - 256;  }  if(checksum == check)  {  //draw box  fill(0,255,0);  rect(490,10,200, 100);  fill(0);  text("Node: " + ID, 510, 40);  text(val + " C", 510, 100);  output.print(month() + "/" + day() + "/" + year() + "/" + hour() + ":" + minute() + ":" + second() + "," + millis());  output.println(",,,,," + val);  }  break;  }  default:  {  }  }  if(key=='x')  {  output.close();  exit();  }  }  void keyPressed()  {  println("key pressed");  if(key=='x')  {  exit();  }  }  void stop()  {  output.close();  super.stop();  } |

**References:**

1. Willig, Andreas; Karl, Holger. *Protocols and Architectures for Wireless Sensor Networks*.
2. Kosmerchock. “Wireless Sensor Network Topologies”.