7. REFERENCES

[1] Theodore S. Rappaport, Wireless communications, 2nd ed, Prentice Hall India, 2004

[2] G. Aggelou, "Multihop Relaying: Stepping Stone to System Beyond 3G" in Mobile Ad Hoc Network from wireless LANs to 4G Networks, Tata McGraw-Hill Edition, New York: 2009, pp. 304-306.

[3] Sangeetha C P, C. D. Suriyakala, "Performance Analysis of IEEE 802.15.4/ZigBee Sensor Networks using ADAPT Algorithm" Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Paper, 2014

[4] A. Setiawan, Farhad Shahnia, Sumedha Rajakaruna and Arindam Ghosh, "ZigBee-Based Communication System for Data Transfer Within Future Microgrids Made" IEEE Transactions on Smart Grid Vol 6, Journal, 2015

[5] Presentation "IEEE Std. 802.15.4" by Dr. Jose A. Gutierrez, Technology Manager, Embedded System & Communication Group

[6] Taehong Kim, Seong Hoon Kim, Jinyoung Yang, Seong-eun Yoo, "Neighbor Table Based Shortcut Tree Routingin ZigBee Wireless Networks" Ieee Transactions On Parallel And Distributed Systems, Vol. 25, No. 3, Journal, March 2014

[7] "What is ZigBee Technology, Architecture and its Applications?", ElProCus - Electronic Projects for Engineering Students, 2014. [Online]. Available: https://www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications/. [Accessed: 18- Apr- 2016].

[8] XBee/XBee-Pro RF Modules, Digi International Inc.

[9] F. M. Sallabi, A. M. Gaouda, A. H. El-Hag, and M. M. A. Salama, "Evaluation of ZigBee wireless sensor networks under high power disturbances," IEEE Trans. Power Del., vol. 29, no. 1, pp. 13–20, Feb. 2014.

[10] C. Tseng, "Coordinator traffic diffusion for data-intensive Zigbee transmission in real-time electrocardiography monitoring," IEEE Rev. Biomed. Eng., vol. 60, no. 12, pp. 3340–3346, Dec. 2013.

[11] Khaldoun Al Agha, Marc-Henry Bertin, Tuan Dang, Alexandre Guitton, Pascale Minet, Thierry Val, and Jean-Baptiste Viollet, "Which Wireless Technology for Industrial Wireless Sensor Networks?" IEEE Trans On Industrial Electronics, Vol. 56, No. 10, October 2009

[12] Pradhumna L. Shrestha, Michael Hempel, Hamid Sharif and Hsiao-Hwa Chen "Modeling Latency and Reliability of Hybrid Technology Networking" IEEE Sensors Journal, Vol. 13, No. 10, October 2013

[13] Junghee Han "Global Optimization of ZigBee Parameters for End-to-End Deadline Guarantee of Real-Time Data" IEEE Sensors Journal, Vol. 9, No. 5, May 2009

[14] Yuan-Yao Shih, Wei-Ho Chung, Pi-Cheng Hsiu, and Ai-Chun Pang, "A Mobility-Aware Node Deployment and Tree Construction Framework for ZigBee Wireless Networks" IEEE Transactions On Vehicular Technology, Vol. 62, No. 6, July 2013

[15] "Wikipedia", Wikipedia.org, 2016. [Online]. Available: https://www.wikipedia.org/. [Accessed: 10- Mar- 2016].

[16] JH Franz, VK Jain, Optical Communication, Narosa, 2006

[17] H Ganapathy Hebbar, Optical Fiber Communicaiton, Fillip Learning India, 2014

[18] "Calculation and usage of LQI and RSSI - Design Notes - Wireless Connectivity - TI E2E Community", E2e.ti.com, 2016. [Online]. Available: http://e2e.ti.com/support/wireless_connectivity/w/design_notes/calculation-and-usage-of-lqi-and-rssi. [Accessed: 22- May- 2016].

APPENDIX A : Meter Readout data

<STX>0.0.0(000147000002)<_CR><_LF>

0.9.2(160624)<_CR><_LF>

0.9.1(160511)<_CR><_LF>

*1.8.0(000004.94*kWh)<_CR><_LF>*

*1.8.0*01(000004.94*kWh)<_CR><_LF>*

*1.8.0*02(000000.90*kWh)<_CR><_LF>*

*1.8.0*03(000000.88*kWh)<_CR><_LF>*

*1.8.0*04(000000.88*kWh)<_CR><_LF>*

*1.8.0*05(000000.31*kWh)<_CR><_LF>*

*1.8.0*06(000000.01*kWh)<_CR><_LF>*

*1.8.0*07(000000.01*kWh)<_CR><_LF>*

*1.8.0*08(000000.01*kWh)<_CR><_LF>*

*1.8.0*09(000000.01*kWh)<_CR><_LF>*

*1.8.0*10(000000.01*kWh)<_CR><_LF>*

*1.8.0*11(000000.00*kWh)<_CR><_LF>*

*1.8.0*12(000000.00*kWh)<_CR><_LF>*

*1.8.1(000001.13*kWh)<_CR><_LF>*

*1.8.1*01(000001.13*kWh)<_CR><_LF>*

*1.8.1*02(000000.13*kWh)<_CR><_LF>*

*1.8.1*03(000000.13*kWh)<_CR><_LF>*

*1.8.1*04(000000.13*kWh)<_CR><_LF>*

*1.8.1*05(000000.07*kWh)<_CR><_LF>*

*1.8.1*06(000000.00*kWh)<_CR><_LF>*

*1.8.1*07(000000.00*kWh)<_CR><_LF>*

*1.8.1*08(000000.00*kWh)<_CR><_LF>*

*1.8.1*09(000000.00*kWh)<_CR><_LF>*

*1.8.1*10(000000.00*kWh)<_CR><_LF>*

*1.8.1*11(000000.00*kWh)<_CR><_LF>*

*1.8.1*12(000000.00*kWh)<_CR><_LF>*

*1.8.2(000001.30*kWh)<_CR><_LF>*

*1.8.2*01(000001.30*kWh)<_CR><_LF>*

*1.8.2*02(000000.28*kWh)<_CR><_LF>*

*1.8.2*03(000000.28*kWh)<_CR><_LF>*

*1.8.2*04(000000.28*kWh)<_CR><_LF>*

*1.8.2*05(000000.08*kWh)<_CR><_LF>*

*1.8.2*06(000000.01*kWh)<_CR><_LF>*

*1.8.2*07(000000.01*kWh)<_CR><_LF>*

*1.8.2*08(000000.01*kWh)<_CR><_LF>*

*1.8.2*09(000000.01*kWh)<_CR><_LF>*

*1.8.2*10(000000.01*kWh)<_CR><_LF>*

*1.8.2*11(000000.00*kWh)<_CR><_LF>*

*1.8.2*12(000000.00*kWh)<_CR><_LF>*

*1.8.3(000001.32*kWh)<_CR><_LF>*

*1.8.3*01(000001.32*kWh)<_CR><_LF>*

*1.8.3*02(000000.31*kWh)<_CR><_LF>*

*1.8.3*03(000000.30*kWh)<_CR><_LF>*

*1.8.3*04(000000.30*kWh)<_CR><_LF>*

*1.8.3*05(000000.07*kWh)<_CR><_LF>*

*1.8.3*06(000000.00*kWh)<_CR><_LF>*

*1.8.3*07(000000.00*kWh)<_CR><_LF>*

*1.8.3*08(000000.00*kWh)<_CR><_LF>*

*1.8.3*09(000000.00*kWh)<_CR><_LF>*

*1.8.3*10(000000.00*kWh)<_CR><_LF>*

*1.8.3*11(000000.00*kWh)<_CR><_LF>*

*1.8.3*12(000000.00*kWh)<_CR><_LF>*

*1.8.4(000001.17*kWh)<_CR><_LF>*

*1.8.4*01(000001.17*kWh)<_CR><_LF>*

*1.8.4*02(000000.16*kWh)<_CR><_LF>*

*1.8.4*03(000000.16*kWh)<_CR><_LF>*

*1.8.4*04(000000.15*kWh)<_CR><_LF>*

*1.8.4*05(000000.07*kWh)<_CR><_LF>*

*1.8.4*06(000000.00*kWh)<_CR><_LF>*

*1.8.4*07(000000.00*kWh)<_CR><_LF>*

*1.8.4*08(000000.00*kWh)<_CR><_LF>*

*1.8.4*09(000000.00*kWh)<_CR><_LF>*

*1.8.4*10(000000.00*kWh)<_CR><_LF>*

*1.8.4*11(000000.00*kWh)<_CR><_LF>*

*1.8.4*12(000000.00*kWh)<_CR><_LF>*

*32.7.0(232.0*V)<_CR><_LF>*

*31.7.0(0.000*A)<_CR><_LF>*

*14.7.0(50.00*Hz)<_CR><_LF>*

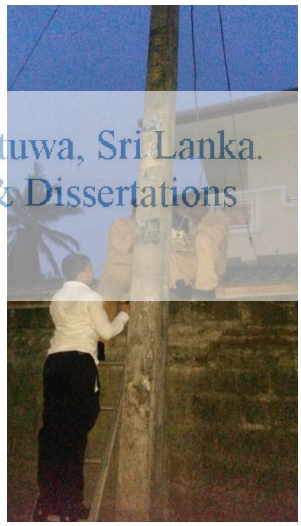
13.7.0(0.000)<_CR><_LF>

F.F.0(00000000)<_CR><_LF>

!<_CR><_LF>

<ETX>L

APPENDIX B : Some Photos



APPENDIX C : Coordinator Program

```
//Zigbee side
//Sending Data
byte StartDelimiter = 0x7E;
byte Length[2];
byte FrameDataTXReq[150];//over estimated
byte SizeOfFrameDataTXReq=0;
byte APIFrame[200];//over estimated
byte SizeOfAPIFrame=0;
byte FrameData[150];//over estimated
byte SizeOfFrameData=0;


//Receiving Data
byte ReceAdd[8];
byte ReceData[100];
byte SizeOfReceData;

//Others
byte FrameID=1;

//Modem side

#include <SoftwareSerial.h>

SoftwareSerial ModemSerial(50, 52); // RX, TX



void setup()
{
  Serial.begin(9600);//for Zigbee side communication
```

```
  ModemSerial.begin(9600);//For modem side communication

}

void loop()
{
  delay(100);
  //reading modem side
  //Serial.write(55);
  if (ModemSerial.available())
  {
            }
            GetAPIFrameTXReq(DesAdd, RFData,SizeOfRFData);
            Serial.write(APIFrame, SizeOfAPIFrame);
          }

        }
       }
     }
    }
  }
}


int GetFrameDataTXReq(byte FrmID, byte DesAdd[], byte RFData[],byte SizeOfRFSData)
{
  int Success=0;
  int i;
  SizeOfFrameDataTXReq=0;
  byte tempFrameDataTXReq[1 + 1 + 8 + 2 + 1 + 1 + SizeOfRFSData];//Length of
frameType FrameID 64DesAdd 16DesAdd BrodRadius Option RFData
  tempFrameDataTXReq[0]=0x10;//frameType
  tempFrameDataTXReq[1]=FrmID;//FrameID
  for(i=0;i<8;i++)//64DesAdd
```

```
 {
  tempFrameDataTXReq[2+i]=DesAdd[i];
 }
 tempFrameDataTXReq[10]=0xFF;//16DesAdd
 tempFrameDataTXReq[11]=0xFE;//16DesAdd
 tempFrameDataTXReq[12]=0;//BrodRadius
 tempFrameDataTXReq[13]=0;//Option
 for(i=0;i<SizeOfRFSData;i++)//RFData
 {
  tempFrameDataTXReq[14+i]=RFData[i];
 }


 SizeOfFrameDataTXReq=sizeof(tempFrameDataTXReq);
 for(i=0;i<SizeOfFrameDataTXReq;i++)
 {
  FrameDataTXReq[i]= tempFrameDataTXReq[i];
 }
```



```
 Success=1;
 return Success;
}
byte GetCheckSum()//done
{
 unsigned long ChkSumLong=0xFF;
 byte ChkSum=0;
 //Serial.write(SizeOfFrameData);
 for(int i=0;i<SizeOfFrameData;i++)
 {
  ChkSumLong -= FrameData[i];
  //Serial.write(FrameData[i]);
```

```
  }
  //Serial.write('A');
  ChkSum=ChkSumLong;
  return ChkSum;
}
byte GetCheckSum(byte Data[],byte SizeOfData)//for modem side
{
  unsigned long ChkSumLong=0xFF;
  byte ChkSum=0;
  //Serial.write(SizeOfFrameData);
  for(int i=0;i<SizeOfData;i++)
  {
    ChkSumLong -= Data[i];
    //Serial.write(FrameData[i]);
  }
  //Serial.write('A');
  ChkSum=ChkSumLong;
  return ChkSum;
}
unsigned int GetLength()//done
{
  unsigned int Length=SizeOfFrameData;
  return Length;
}

int GetAPIFrameTXReq(byte DesAdd[], byte RFData[],byte SizeOfRFData)
{
  int Success=0;
  SizeOfAPIFrame=0;
  int i;
//  FrameDataTXReq={0,0};//Initialize the FrameDataTXReq
  //Serial.write(1);
  GetFrameDataTXReq(FrameID, DesAdd, RFData,SizeOfRFData);
  //Serial.write(FrameDataTXReq, SizeOfFrameDataTXReq);
```

```
 SizeOfFrameData=SizeOfFrameDataTXReq;
 for(i=0;i<SizeOfFrameData;i++)
 {
  FrameData[i]=FrameDataTXReq[i];
 }



 byte CheckSum = GetCheckSum();
// Serial.write(2);Serial.write(CheckSum);
 unsigned int tempLength = GetLength();
//Serial.write(3);
 Length[0]=tempLength;
 Length[1]=tempLength >> 8 ;
//  Serial.write(4);
 byte tempAPIFrame[1+2+SizeOfFrameDataTXReq + 1];//length of StartDelimiter length
FrameData CheckSum
//  Serial.write(5);

 tempAPIFrame[0]=StartDelimiter;//StartDelimiter
//  Serial.write(6);
 tempAPIFrame[1]=Length[1];//length
 tempAPIFrame[2]=Length[0];//length
// Serial.write(7);
 for(i=0;i<SizeOfFrameData;i++)
 {
  tempAPIFrame[i+3]=FrameData[i];
 }
 tempAPIFrame[SizeOfFrameDataTXReq+2+1]=CheckSum;

 SizeOfAPIFrame=sizeof(tempAPIFrame);
 for(i=0;i<SizeOfAPIFrame;i++)
 {
```

```
  APIFrame[i]=tempAPIFrame[i];
 }
 //Serial.write(APIFrame, SizeOfAPIFrame);



 FrameID ++;
 Success=1;
 return Success;
}

boolean CheckCheckSum(char bufferWithCSum[], byte SizeOfbufferWithCSum)//Checking
checksum for receiving data
{
 boolean Success=false;
 int IntSum=0;
 byte Sum=0;
 for(int i=0;i<SizeOfbufferWithCSum;i++)
 {
  IntSum +=(byte)bufferWithCSum[i];
 }
 Sum=IntSum;
 if(Sum == 0xFF)
  Success=true;
 return Success;
}

boolean CheckCheckSum(char buffer1[], byte SizeOfbuffer1, char buffer2WithCSum[], byte
SizeOfbuffer2WithCSum)//Checking checksum for receiving data
{
 boolean Success=false;
 int IntSum=0;
 byte Sum=0;
 for(int i=0;i<SizeOfbuffer1;i++)
```

```
 {
  IntSum +=(byte)buffer1[i];
 }
 for(int i=0;i<SizeOfbuffer2WithCSum;i++)
 {
  IntSum +=(byte)buffer2WithCSum[i];
 }
 Sum=IntSum;
 if(Sum == 0xFF)
  Success=true;
 return Success;
}


void serialEvent() //for zigbee side serial
{
 if(Serial.available())
 {
  if (Serial.read() == 0x7E)//checking Start delimiter
  {
      char bufferLen[2]; //reading length
      Serial.setTimeout(1000);
      if(Serial.readBytes(bufferLen, 2)!=0)
      {
        int length= (((int)bufferLen[0]) << 8) + (int)bufferLen[1];
        char bufferData[length+1];
        Serial.setTimeout(1000);
        if(Serial.readBytes(bufferData, length + 1)!=0)//Reading Frame data +1 for
checksum
        {
          if(bufferData[0] == (char)0x90)//Receive Packet
          {
            if(CheckCheckSum(bufferData, length + 1)==true)//error in checksum
            {
```

```
for(int i=0;i<8;i++)
{
  ReceAdd[i]=bufferData[i+1];
}
SizeOfReceData=length-12;
for(int i=0;i<SizeOfReceData;i++)
{
  ReceData[i]=bufferData[i+12];
}

//arrange data for sending to modem
byte ReceAddAndData[8+SizeOfReceData];

for(int i=0;i<8;i++)
{
  ReceAddAndData[i]=(byte)ReceAdd[i];
}
for(int i=8;i<sizeof(ReceAddAndData);i++)
{
  ReceAddAndData[i]=(byte)ReceData[i-8];
}

//Sending data to modem
ModemSerial.write(0x7E);//start delimiter
int LengthReceAddAndData=sizeof(ReceAddAndData);
byte Length[2];
Length[0]=LengthReceAddAndData;
Length[1]=LengthReceAddAndData >> 8 ;
for(int i=0;i<8;i++)
{
  ModemSerial.write(ReceAddAndData[i]);//length
}
ModemSerial.write(Length[1]);
```

```
            ModemSerial.write(Length[0]);
            for(int i=8;i<sizeof(ReceAddAndData);i++)
            {
              ModemSerial.write(ReceAddAndData[i]);//length
            }

ModemSerial.write(GetCheckSum(ReceAddAndData,sizeof(ReceAddAndData)));//check sum
          }
        }
      }
    }


  }
 }
}
```

APPENDIX D : LQI Variation

APPENDIX E : Simulation Program-Communication Speed

Function Distance Matrix

```
function DisMtrix = fcn()
%#codegen
L=[1 2;%Locations
   2 3;
   4 5;
   5 6;
   1 2;
   2 3;
   4 5;
   5 6;
   4 5;
   5 6
   ];
m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10

      m(r,c) =  (sqrt((L(r,1)-L(c,1))^2 + (L(r,2)-L(c,2))^2));
   end;

end;
DisMtrix=m
```


Function Signal Strength

```
function y = SigStr(Distace)
%#codegen

m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10

      m(r,c) =  1/(Distace(r,c))^2;
   end;

end;
y=m
```

Function Modify Noise

```
function xx = fcn(u)
%#codegen
mm= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10
      if c<r
```

```
      mm(r,c) =  u(r,c);
      mm(c,r) =  u(r,c);
      end ;

   end;

end;
xx = mm;
```

Function Communication Matrix

```
function CommunicationMtx = fcn(SigStrCutLmt,SigStrMtx)
%#codegen
m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10
      if SigStrMtx(r,c)>SigStrCutLmt

          m(r,c)=1;



      else

          m(r,c)=0;
      end;
   end;

end;
CommunicationMtx=m
```


Function Communication Speed

```
function ComSpeed = ComSp(SS,RN)
%#codegen

m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      m(r,c) =9600*log2(1+(SS(r,c)/ RN(r,c)));

   end;

end;
ComSpeed=m;
```

Function Communication links

```
function SpeedsOfLinks = fcn(ComMatrix,ComSpeed)
%#codegen

m= zeros(10,10);
for r = 1:10;

   for c=1:10;
       if ComMatrix(r,c)==1
           m(r,c)=ComSpeed(r,c)
       end
   end;

end;

SpeedsOfLinks =m;
```

Function Low Speed Links

```
function LowSpeedLinks = fcn(SpeedsOfLinks,SpeedRef)
%#codegen

m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10
      if SpeedsOfLinks(r,c)<SpeedRef
          m(r,c)=1;
      else
          m(r,c)=0;
      end;
   end;


end;

LowSpeedLinks = m;
```


Function Reroute

```
function NewComSpeed = fcn(ComSpeed,LowSpeedLinks,ComMatrix)
%#codegen


m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10
      if LowSpeedLinks(r,c)==1
          for cc = 1:10;
              if ComMatrix(r,cc)==1
                  ComSpeed(r,cc)=ComSpeed(r,cc)*98/100
              end;
```

```
            end;

            for rr=1:10;
                if ComMatrix(rr,c)==1
                    ComSpeed(rr,c)=ComSpeed(r,cc)*98/100
                end;
            end;
        end;
    end;


end;


NewComSpeed = ComSpeed;
```

APPENDIX F : Simulation Program-Communication Path

Funuction Distance Matrix

```
function DisMtrix = fcn()
%#codegen
L=[1 1;%Locations
   2 4;
   4 3;
   7 1;
   12 1;
   13 8;
   7 7;
   2 11;
   7 11;
   12 11
   ];
m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10

      m(r,c) =  (sqrt((L(r,1)-L(c,1))^2 + (L(r,2)-L(c,2))^2));
   end;

end;
DisMtrix=m
```


Function Signal Strength

```
function y = SigStr(Distace)
%#codegen

m= zeros(10,10);
for r = 1:10;

   for c=1:10;
      % m(r,c)=10

      m(r,c) =  100/(Distace(r,c))^2;
   end;

end;
y=m
```

Function Communication Matrix

```
function CommunicationMtx = fcn(SigStrCutLmt,SigStrMtx)
%#codegen
m= zeros(10,10);
for r = 1:10;
```

```
    for c=1:10;
       % m(r,c)=10
       if SigStrMtx(r,c)>SigStrCutLmt

           m(r,c)=1;



       else

           m(r,c)=0;

       end;
    end;


end;
CommunicationMtx=m
```

Function Current Position

```
function Out = fcn()
%#codegen
Target=[4 9 5 ]

Out = Target;
```


Function Simulation

```
function [Path,Durations]=
fcn(ComMatrix,Distance,CurrentPositionArray,Taget)
%#codegen
ComMatrixSize=[10 10];
DistanceMatrixSize=ComMatrixSize;
CurrentPositionSize=3;
CurrentPositionHistory = zeros(CurrentPositionSize,100);
OccupiedNodes=[0 0 0 0 0 0 0 0 0 0];
for i=1:CurrentPositionSize;
    CurrentPositionHistory(i,1)=CurrentPositionArray(i);

end;

for Time=2:100;
for i=1:CurrentPositionSize;
    if CurrentPositionArray(i)==1
    else
        OccupiedNodes(CurrentPositionArray(i))=1;
    end;

end;
```

```
for CurrentPositionIndex = 1:CurrentPositionSize;


CurrentPositionHistory(CurrentPositionIndex,Time)=CurrentPositionArr
ay(CurrentPositionIndex);

SelectedCurrentPosition=CurrentPositionArray(CurrentPositionIndex);
  if SelectedCurrentPosition == 1

  else

  SelectedLowestDistance=1000;%Temp value only
  SelectedNode=0;
  for ComIndex=1:ComMatrixSize(2);
      if ComMatrix(SelectedCurrentPosition,ComIndex)==1
          if Distance(ComIndex,1) < SelectedLowestDistance
              if OccupiedNodes(ComIndex) == 0
                SelectedLowestDistance=Distance(ComIndex,1);
                SelectedNode=ComIndex;
              end;
          end;
      end;

  end;
  if SelectedNode > 0

CurrentPositionHistory(CurrentPositionIndex,Time)=SelectedNode;
      CurrentPositionArray(CurrentPositionIndex)=SelectedNode;
      OccupiedNodes(SelectedNode)=1;
  end;
end ;
```



```
  % ComMatrix
  %Path=1;
   %for c=1:10;
      % m(r,c)=10

     % m(r,c) =  (sqrt((L(r,1)-L(c,1))^2 + (L(r,2)-L(c,2))^2));
   %end;

end;
OccupiedNodes=[0 0 0 0 0 0 0 0 0 0];
end;
%Path = Target;
Path=CurrentPositionHistory
Durations=2;
```

APPENDIX G : Program-VS-Finding Levels and Zigbee pro selection

```
public bool[] IsZigbeePro;
        public int[] NoOfComLinks;
        public int NoOfLinkMinLimit;
        public NetworkData()
        {
            IsZigbeePro = new bool[Coordinates.GetLength(0)];
            for (int i = 0; i < IsZigbeePro.Length; i++)
            {
                IsZigbeePro[i] = true;
            }

            CommunicationMatrix = new
bool[CommunicationMatrixInt.GetLength(0),
CommunicationMatrixInt.GetLength(0)];
            for (int r = 0; r < CommunicationMatrixInt.GetLength(0);
r++)
            {
                for (int c = 0; c <
CommunicationMatrixInt.GetLength(0); c++)
                {
                    CommunicationMatrix[r, c] =
CommunicationMatrixInt[r, c] == 1 ? true : false;
                }
            }
        }
        public NetworkData(bool IsAllZigbeePro,int NodeCount)
        {
            IsZigbeePro = new bool[NodeCount];
            for (int i = 0; i < IsZigbeePro.Length; i++)
            {
                IsZigbeePro[i] = IsAllZigbeePro;
            }
            NoOfComLinks = new int[NodeCount];

            CommunicationMatrix = new
bool[CommunicationMatrixInt.GetLength(0),
CommunicationMatrixInt.GetLength(0)];
            for (int r = 0; r < CommunicationMatrixInt.GetLength(0);
r++)
            {
                for (int c = 0; c <
CommunicationMatrixInt.GetLength(0); c++)
                {
                    CommunicationMatrix[r, c] =
CommunicationMatrixInt[r, c] == 1 ? true : false;
                }
            }
        }
        public double[,] GetDistanceMatrix(double[,] Codints)
        {
            int CoordinatesLength = Codints.GetLength(0);
            double[,] DistanceMatrix = new double[CoordinatesLength,
CoordinatesLength];
            for (int r = 0; r < CoordinatesLength; r++)
            {
                for (int c = 0; c < CoordinatesLength; c++)
```

```
                {
                    DistanceMatrix[r, c] =
Math.Sqrt(Math.Pow((Codints[r, 0] - Codints[c, 0]), 2) +
Math.Pow((Codints[r, 1] - Codints[c, 1]), 2));
                }
            }
            return DistanceMatrix;
        }
        public bool[,] GetCommunicationMatrix(double[,]
DistanceMatrix, bool[] IsZigbeePro)
        {
            int Count=IsZigbeePro.Length;
            bool[,] ComMatrix = new bool[Count, Count];
            for (int r = 0; r < Count; r++)
            {
                for (int c = 0; c < Count; c++)
                {
                    if(IsZigbeePro[r] & IsZigbeePro[c])
                    {
                        if(DistanceMatrix[r, c]<ZigbeeProComDis)
                        {
                            ComMatrix[r, c]=true;
                        }
                        else
                        {
                            ComMatrix[r, c]=false;
                        }
                    }
                    else
                    {
                        if(DistanceMatrix[r, c]<ZigbeeComDis)
                        {
                            ComMatrix[r, c]=true;
                        }
                        else
                        {
                            ComMatrix[r, c]=false;
                        }
                    }
                }
            }
            return ComMatrix;
        }
        public int UpdateNoOfLinks()
        {
            int MinNoOfLinks = Coordinates.GetLength(0);
            for (int r = 0; r < Coordinates.GetLength(0); r++)
            {
                int TempNoOfLinks = 0;
                for (int c = 0; c < Coordinates.GetLength(0); c++)
                {
                    if (CommunicationMatrix[r, c])
                    {
                        TempNoOfLinks++;
                    }
                }
                NoOfComLinks[r] = TempNoOfLinks;
                if (MinNoOfLinks > TempNoOfLinks)
```

```
                {
                    MinNoOfLinks = TempNoOfLinks;
                }
            }
            return MinNoOfLinks;
        }
        public void UpdateProModule()
        {
            try
            {
                int TempMinNoOfLinks =
Coordinates.GetLength(0);//Temp Value
                int TempMinNoOfLinksIndex=-1;
                bool[] TempIsChecked = new
bool[Coordinates.GetLength(0)];
                for (int rr = 0; rr < Coordinates.GetLength(0);
rr++)
                {
                    TempMinNoOfLinks =
Coordinates.GetLength(0);//Temp Value
                    int r = rr;
                    for (int tempr = 0; tempr <
Coordinates.GetLength(0); tempr++)
                    {
                        if ((TempMinNoOfLinks >
NoOfComLinks[tempr])&&(! TempIsChecked[tempr]))
                        {
                            TempMinNoOfLinksIndex = tempr;
                            TempMinNoOfLinks = NoOfComLinks[tempr];
                            r = tempr;
                        }
                    }
                    TempIsChecked[r] = true;

                    if (NoOfComLinks[r] < NoOfLinkMinLimit)
                    {
                        if (!IsZigbeePro[r])
                        {
                            IsZigbeePro[r] = true;
                            break;
                        }
                        else
                        {
                            double tempDisMin = 5000;//temp max
value
                            int tempDisMinIndex = -1;
                            for (int c = 0; c <
Coordinates.GetLength(0); c++)
                            {
                                if ((tempDisMin > DistanceMatrix[r,
c]) && (!IsZigbeePro[c]))
                                {
                                    tempDisMin = DistanceMatrix[r,
c];
                                    tempDisMinIndex = c;
                                }
                            }
                            if (tempDisMin <= ZigbeeProComDis)
```

```
                                {
                                    IsZigbeePro[tempDisMinIndex] = true;
                                    break;
                                }

                            }

                        }
                    }
                }
                catch
                {
                }
            }



public class NetworkCombinations
    {
        bool[] OccupiedList;
        bool[,] ComMatrix;
        public NetworkCombinations(bool[,] CommunicationMatrix)
        {
            bool[] OccupiedList=new
bool[CommunicationMatrix.GetLength(0)];
            ComMatrix = CommunicationMatrix;
        }
        public bool[] GetCombinations(bool[] ParentCombinations,ref
bool[] OccupiedList,bool[,] CommunicationMatrix)

            bool[] ChildList=new
bool[CommunicationMatrix.GetLength(0)];
            for (int i = 0; i < ParentCombinations.GetLength(0);
i++)
            {
                if (ParentCombinations[i])
                {
                    int r = i;
                    for (int c = 0; c <
CommunicationMatrix.GetLength(1); c++)
                    {
                        if (CommunicationMatrix[r, c] == true)
                        {
                            if (!OccupiedList[c])
                            {
                                ChildList[c] = true;
                                OccupiedList[c] = true;
                            }
                        }
                    }
                }

            }
            return ChildList;

        }
```

```
        public ArrayList GetCombinationsInLevels()
        {
            bool[] ChildList;
            bool[] ParentCombinations=new
bool[ComMatrix.GetLength(0)];
            ParentCombinations[1]=true;
            OccupiedList=new bool[ComMatrix.GetLength(0)];
            ArrayList Levels = new ArrayList();
            Levels.Add(ParentCombinations);
            while (OccupiedList.Contains(false))
            {
                ChildList = GetCombinations(ParentCombinations, ref
OccupiedList, ComMatrix);
                ParentCombinations = ChildList;
                Levels.Add(ParentCombinations);
            }
            return Levels;
        }


    }
```

APPENDIX H : BZ 501 Transformer area map