**A Mini Project Report on**

**R.E.A.S.N – Redundancy Elimination Algorithm for Sensor Network**

**Submitted By**

**Shobhan Nath**

**Shivani Bhandari**

**Shashank Sharma**

A Mini Project report submitted as a partial fulfillment towards the Term- V of

***T.E. (Electronics & Telecommunication)***

Savitribai Phule Pune University, Pune.



Department of E&TC Engineering

Army Institute of Technology, Dighi, Pune-411 015

**2018-19**

**1**  **INTRODUCTION**

1.1 Problem Definition/identification of project/objectives of project

1.2 Historical Background

**2** **BLOCK DIAGRAM / DESCRIPTION/ FLOW CHART /**

**ALGORITHM**

**3** **PROJECT SPECIFICATION AND CIRCUIT DIAGRAM**

**4 PROJECT REQUIREMENTS**

**5.1 HARDWARE REQUIREMENT**

**5.2 SOFT WARE REQUIREMENT**

**5 LIST OF COMPONENTS USED**

**6** **COMPONENT DESCRIPTION**

**7. SOURCE CODE**

**8. LIST COMPONENTS WITH TOTAL COST**

**CONCLUSION**

**REFERENCES** (AS PER IEEE FORMAT FONT SIZE 12)

**APPENDIX** (IF ANY)

**ABSTRACT**

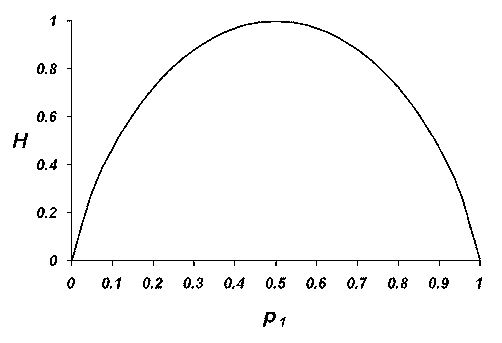
Electronics component manufacturers and circuit designers from all around the globe are currently short of two things, time and space (memory). Most of the time in electronic circuit designing we have abundant time but same thing can’t be said about space.

Microcontrollers have advanced by leaps in the last 20 years, in almost all regards, higher clock speed, more peripherals and easier debugging. But it is still common to see RAM in the 10’s of Kilobytes.

Hence to implement a case where we are receiving data from multiple channels over a single hub from certain pool of networked sensors but the memory to store this data is available for only a few sensors. So how do we pick which of the sensor is more redundant?

Introducing REASN which is an acronym for *Redundancy Elimination Algorithm for Sensor Network* a project implemented by three B.E undergraduates. REASN architecture is based upon REA which is again an acronym for Redundancy Elimination Algorithm written on the concept of entropy.

To prioritize the sensors on the basis of their environmental characteristics won’t be futile but to characterize them on the basis of their previous and current data along with the conception that information retained from an event whose probability is 1 is 0 and the information retained from an event whose probability should be 0.5 but it still occurred is maximum could make some difference.



**Fig1: Entropy vs. Probability**

**INTRODUCTION**

Sensors are widely used in many different applications and sensor technology has become a basic enabling technology in many instances. The increased use of sensors in smart phones and other consumer electronics is triggering market growth. The high penetration of these sensors in the global wearable devices market as a result of their increased adoption in devices such as smart watches and other fitness devices will further fuel market growth during forecast period

.All sensors transmitting data as simple as proximity, temperature, pressure, smoke concentration and as complex as transmitting live feed of traffic or acoustic data from ocean floor thousands of fathom deep need memory to store this enormous data.

While there has been an exponential development in designing faster algorithms for data fetching and processing still the size of Random Access Memory on a microcontroller is in multiples of 10’s. Hence we have tried to solve the problem using sensor prioritization.

**About Redundancy Elimination Algorithm:**

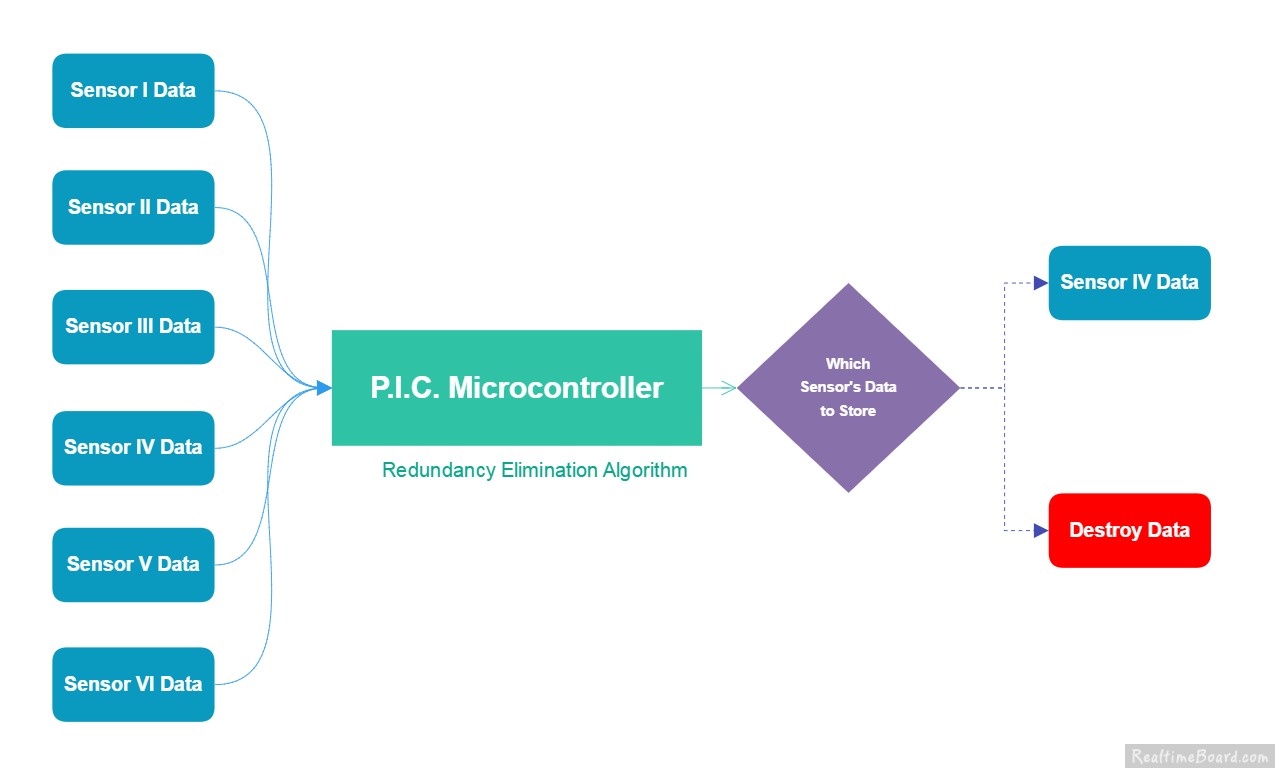
Entropy – Entropy of a source is average amount of information generated by the source per message.

Mathematically the equation of entropy is give by,

Let us consider a sensor source X could (not-known beforehand) generate m different messages x1, x2 …… xm with corresponding probabilities p(x1), p(x2)…. p (xm). It means in a long stream of messages generated by source, the probability of generating message xi will be p (xi).

Likewise the hub or microcontroller gets stream of data from Y, Z and other sources as well. So we calculate the entropy (average information) generated from each individual source and store the information generated from the source generating the maximum information.

**FLOWCHART**

****

**Fig number: 2 Single Node Flow Chart**

**Explanation:**

1. The rate of data transfer from the entire sensor array should be kept constant and passed on to the P.I.C. Microcontroller. Sensors used must send compatible data with each other and with the microcontroller.
2. The P.I.C. Microcontroller used here in the depiction already has the Redundancy Elimination algorithm written within its program hence first step of elimination takes place here.
3. Hence out of all the sensors microcontroller chooses that particular sensor whose data should be stored and sends it to the server. And the remaining sensor’s data is eliminated or destroyed.

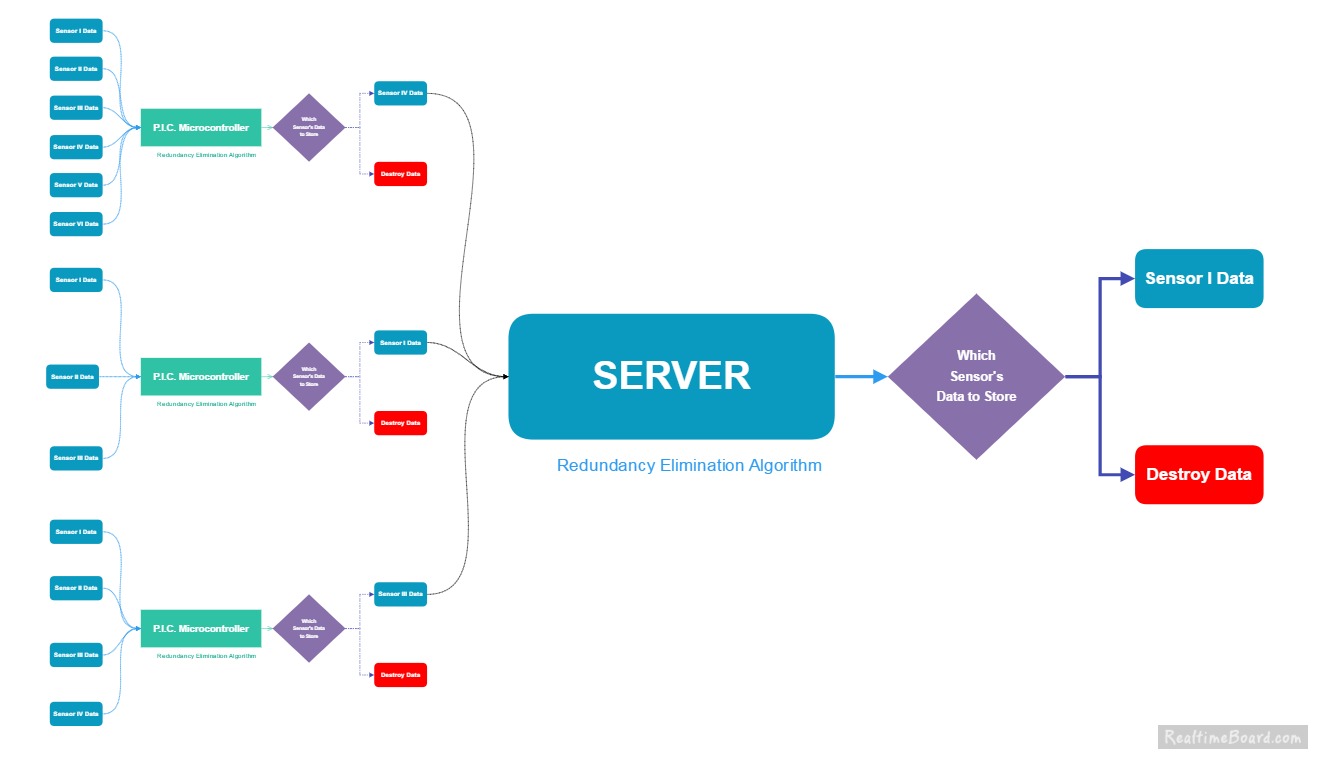


Figure Number: 3 Full Node Flow Chart

**Explanation:**

1. The sensor whose data is prioritized is sent to the server where server keeps the entire sensor’s data from all different location. A further level prioritization is done using R.E.A.
2. Multiple servlets are created with HTML at frontend for form/ data processing needs. The data from the sensor are entered into this form. The data fetched from the sensor is further given as input into Shivani - 89 the Redundancy Elimination Algorithm which maps the inputs on the basis of their count. A probability distribution is calculated from the data provided and further the entropy is determined.
3. The whole servlet runs on the Apache Tomcat Servlet container.

**CIRCUIT DIAGRAM**

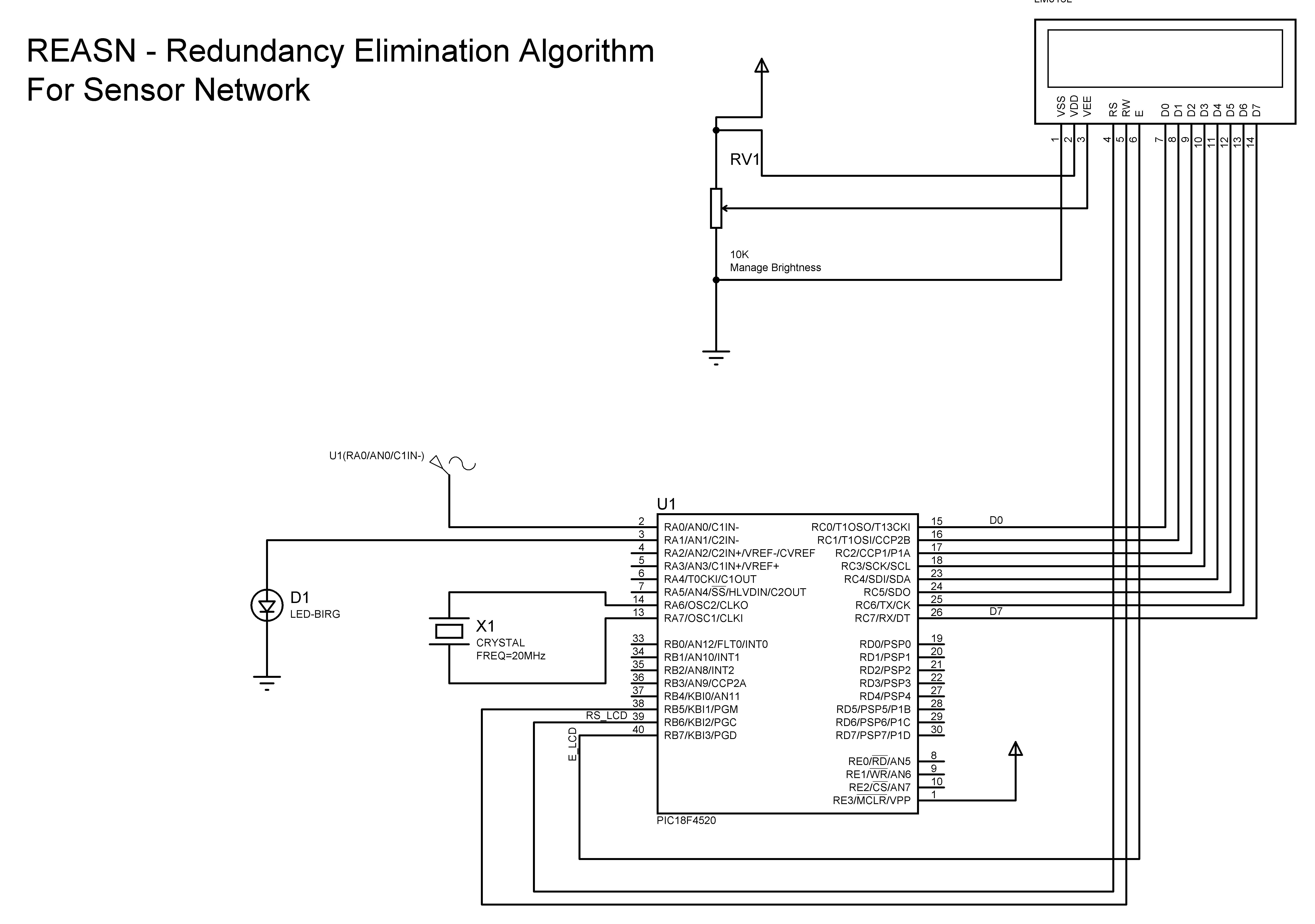


Figure Number 4: Circuit Diagram

**PROJECT SPECIFICATION**

**LIST OF COMPONENTS REQUIRED**

* P.I.C. Microcontroller
* Sensor (MQ 135, MQ 3, F.S.R)
* L.C.D. Display 16x2
* Server
* R.E.A. Algorithm

**COMPONENT REQUIREMENT AND DESCRIPTION**

**PIC MICROCONTROLLER**

**MICROCONTROLLER: PIC 18F4520**

PIC is the name of micro-controllers developed by Microchip Corporation. PIC is the acronym for Peripheral Interface Controller. This chapter will give you an introduction to the world of PIC in general.

Peripheral Highlights

* High-current sink/source 25 mA/25 mA
* Three programmable external interrupts
* Four input change interrupts
* Up to 2 Capture/Compare/PWM (CCP) modules,
* Enhanced Capture/Compare/PWM (ECCP) module
* Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI (all 4 modes) and I2C™ Master and Slave Modes
* Enhanced Addressable USART module:
* 10-bit, up to 13-channel Analog-to-Digital
* Dual analog comparators with input multiplexing

PIC18F devices provides

* High computational performance at an economical price
* 8 x 8 Hardware Multiplier
* Enhanced Flash program memory of sizes from 8Kbto 2Mbytes.
* Data memory from 256byte to 4Kbytes
* Priority Level on interrupts



Fig.6.a.PIC 18F4520

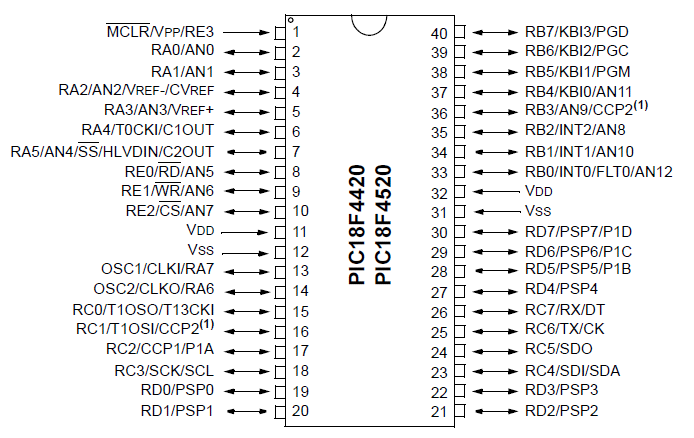


Fig.PIC18F4520 PIN DIAGRAM

**SNS - MQ135**

SNS-MQ135 is a sensitive air quality sensor that detects NH3, NOx, alcohol, benzene, smoke, CO2, etc. They are used in air quality control equipments for buildings/ offices.

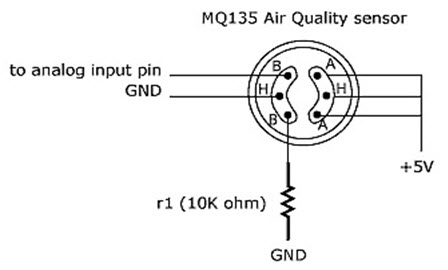
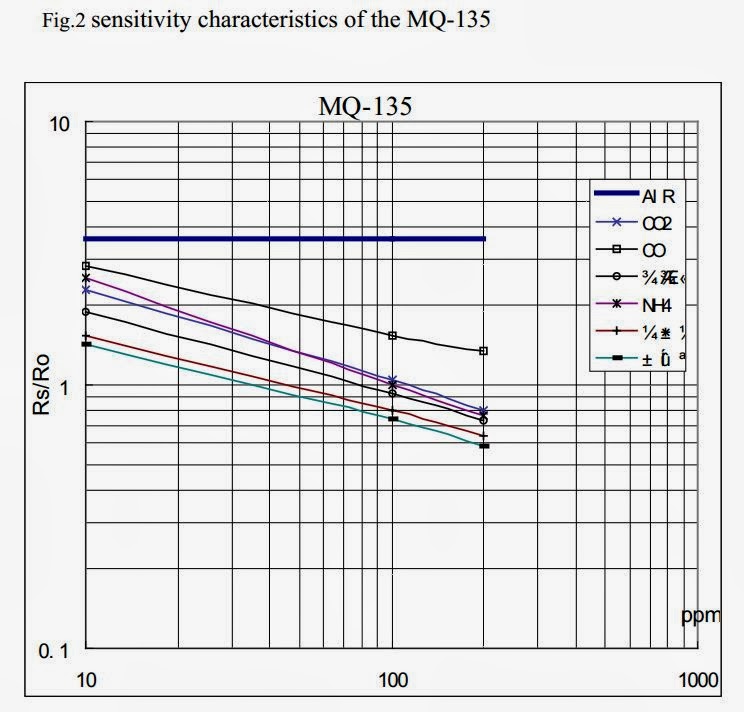


Fig: Configuration of SNS-MQ135

Features of MQ135:

* Wide detecting scope
* Fast response and high sensitivity
* Simple drive circuit
* Stable and long life
* Operating voltage 5V
* Operating current 40mA
* Both digital and analog output

Sensor is composed by micro Al2O3 ceramic tube, Tin dioxide sensitive layer, measuring electrode and heater are fixed into crust made by plastic and stainless steel net. The heater is fixed into crust made by plastic and stainless steel net. The heater provides the necessary work condition for work of sensitive components. The enveloped MQ-135 have 6 pin, 4 of them are used to fetch signals and other 2 are used for providing heating current.

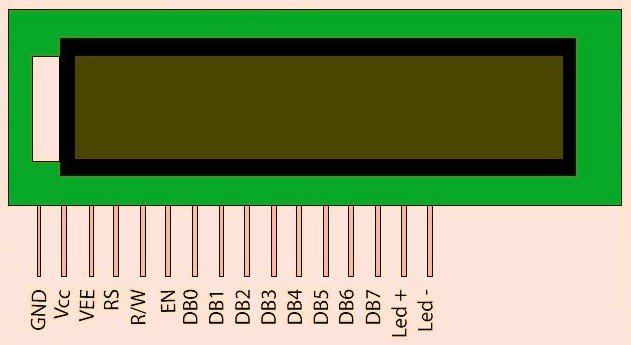


The application of sensor in our project is of utmost importance. Since the project is Omni-sensor and hence does not depend on sensor it is working with, the sensor’s data is crucial to work with. The sensor data is fetched and applied as input to the PIC microcontroller at the first level of elimination/ Node modules and the further elimination is done at the server.

We can try different sensors with the project but the result won’t be affected and will only be judged upon the information it is currently transmitted.

**LIQUID CRYSTAL DISPLAY**

LCD (Liquid Crystal Display) screen is an electronic display module which uses liquid crystal to produce a visible image. The 16x2 module is widely used and hence and find a wide range of applications. It translates and displays 16 characters per line in 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix.



Pin configuration of LCD

1. GND Ground
2. VCC Supply voltage; 5v
3. VEE Contrast adjustment; the best way is to use a variable resistor such as a potentiometer. The output of the potentiometer is connected to this pin. Rotate the potentiometer knob forward and backwards to adjust the LCD contrast.
4. RS – Selects command register when low, and data register when high.
5. R/W – Low to write to the register; High to read from the register.
6. E – Enable to send data to data pins from the microcontroller when a high to low pulse is given.
7. 8 bit data pins from DB0 to DB7
8. Backlight VCC (5V)
9. Backlight Ground (OV)

Important command codes for LCD

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Hex Code** | **Command to LCD instruction Register** |
| 1 | 01 | Clear display screen |
| 2 | 02 | Return home |
| 3 | 04 | Decrement cursor (shift cursor to left) |
| 4 | 06 | Increment cursor (shift cursor to right) |
| 5 | 05 | Shift display right |
| 6 | 07 | Shift display left |
| 7 | 08 | Display off, cursor off |
| 8 | 0A | Display off, cursor on |
| 9 | 0C | Display on, cursor off |
| 10 | 0E | Display on, cursor blinking |
| 11 | 0F | Display on, cursor blinking |
| 12 | 10 | Shift cursor position to left |
| 13 | 14 | Shift cursor position to right |
| 14 | 18 | Shift the entire display to the left |
| 15 | 1C | Shift the entire display to the right |
| 16 | 80 | Force cursor to beginning ( 1st line) |
| 17 | C0 | Force cursor to beginning ( 2nd line) |
| 18 | 38 | 2 lines and 5×7 matrix |

The aim of using a display in our project is to visually represent the data fetching from the sensor and further processing data using in-built ADC of PIC microcontroller. The digital data from the ADC is displayed on LCD with 4 data per line of LCD at an interval of 1 data per second.

**METHODOLOGY**

**WORKING OF PROJECT: STEPS**

1. A single sensor is connected to the PIC microcontroller 16F4520 to pin number 2. Due to unavailability of sensor we divide the sensor output in specific time durations called segments. This duration must be kept constant for each segment since the rate of data fetching is 1 message per second.
2. We take three segments (time-intervals) namely from T1 – T2, T2 – T3 and finally T3 – T4. Each segment represents a separate sensor. The sensor’s data for each segment is stored in the PIC microcontroller’s RAM.
3. The data for each segment is displayed on the LCD screen for an alternative storing of the data i.e. onto the server via the Servlet and Servlet container Apache Tomcat.
4. Each segment’s data is applied as input to the Redundancy Elimination Algorithm. Here the data is compiled and probability of each Shobhan - 78 particular data is calculated.

1. We further calculate the entropy of each sensor based on these data fetched from the sensor and the entropies of each segment is calculated and observed in decreasing order.

1. Except the sensor having maximum entropy we eliminate the data stored from other sensors. We place the sensor with maximum entropy as first priority.

**SOURCE CODE**

#include<p18f4520.h> //to include controller specific header file

#pragma config OSC = HS //Oscillator selection {4, 10, 20, 25 MHz}

#pragma config WDT = OFF //disable watchdog timer

#pragma config LVP = OFF //disable low voltage programming

#pragma config PBADEN = OFF //disable PORT B

//Macros defined for LCD

#define RW PORTBbits.RB1

#define RS PORTBbits.RB0

#define EN PORTBbits.RB2

//function prototypes

void ADC\_Init(void);

unsigned int Get\_ADC\_Result(void);

void ms\_delay(unsigned int time);

void init\_lcd(void);

void lcd\_command(unsigned char cmd);

void lcd\_data(unsigned char data);

//main function

void main()

{

unsigned int result;

unsigned char i,thousands, hundreds, tens, ones;

//input-output port configuration

//declare PORT C as output for LCD

//TRISC = 0x00;

TRISCbits.TRISC0 = 0;

TRISCbits.TRISC1 = 0;

TRISCbits.TRISC2 = 0;

TRISCbits.TRISC3 = 0;

TRISCbits.TRISC4 = 0;

TRISCbits.TRISC5 = 0;

TRISCbits.TRISC6 = 0;

TRISCbits.TRISC7 = 0;

//For RS and E of LCD declared as output

TRISBbits.TRISB6 = 0;

TRISBbits.TRISB7 = 0;

//For sensor input (1 Sensor A)

TRISAbits.TRISA0 = 1;

//For led

TRISAbits.TRISA1 = 0;

//Initialization of A/D and LCD

init\_lcd();

ADC\_Init();

//main code starts here

while(1)

{

result = Get\_ADC\_Result();

i = result/1000 ;

thousands = i + 0x30;

lcd\_data (thousands);

i = (result%1000)/100;

hundreds = i + 0x30;

lcd\_data (hundreds);

i = ((result%1000)%100)/10;

tens = i + 0x30;

lcd\_data (tens);

i = result%10 ;

ones = i + 0x30;

lcd\_data (ones);

ms\_delay(5000); //5 seconds delay

}

}

unsigned int Get\_ADC\_Result()

{

unsigned int ADC\_Res = 0;

ADCON0bits.GO = 1; //start conversion

while(ADCON0bits.GO); //wait till conversion is complete

ADC\_Res = (unsigned int)ADRESH; //Since left justified

return ADC\_Res;

}

void ADC\_Init()

{

//initialize ADC module. It contains 5 important registers

/\*

ADCON0 = bit 7-6 Unimplemented 00 -- bit 5-2 Channel 0 Select 0000 -- bit 1 A/D idle 0 - bit 0 A/D Disabled

\*/

ADCON0 = 0b00000000; //A/D module is OFF and channel 0 is selected

/\*

ADCON1 = bit 7-6 Unimplemented 00 -- bit 5 VSS 0 -- bit 4 VDD 0 -- bit 3-0 Control bits 1110 Select AN0 analog input

\*/

ADCON1 = 0b00001110;

/\*

ADCON2 = bit 7 Left Justified(Copy ADRESH) 0 -- bit 6 Unimplemented 0 -- bit 5-3 2Tad 001 -- bit 2-0 Fosc/64 110

\*/

ADCON2 = 0b00001110;

ADCON0bits.ADON = 1; //A/D Converter module is enabled

}

void init\_lcd(void)

{

lcd\_command(0x38);

ms\_delay(25);

lcd\_command(0x01);

ms\_delay(25);

lcd\_command(0x0C);

ms\_delay(25);

lcd\_command(0x80);

ms\_delay(25);

}

void lcd\_command(unsigned char cmd)

{

PORTC = cmd;

RS = 0;

RW = 0;

EN = 1;

ms\_delay(25);

EN = 0;

}

void lcd\_data(unsigned char data)

{

PORTC = data;

RS = 1;

RW = 0;

EN = 1;

ms\_delay(25);

EN = 0;

}

void ms\_delay(unsigned int time)

{

unsigned int i, j;

for(i = 0;i<time;i++)

for(j = 0;j<275;j++); //calibrated for 1ms in MPLAB

}

**SOURCE CODE FOR REDUNDANCY ELIMINATION ALGORITHM**

REA.JAVA

**package** main;

**import** java.util.HashMap;

**public** **class** REA\_Hub

{

**private** String sensor\_data1;

**private** String sensor\_data2;

**private** String sensor\_data3;

**public** REA\_Hub(String s1, String s2, String s3)

{

**this**.sensor\_data1 = s1;

**this**.sensor\_data2 = s2;

**this**.sensor\_data3 = s3;

}

**public** **int** getHighestPrioritySensor()

{

**int**[] Sensor1\_DataSet = getDataSet(sensor\_data1);

**int**[] Sensor2\_DataSet = getDataSet(sensor\_data2);

**int**[] Sensor3\_DataSet = getDataSet(sensor\_data3);

**double** Sensor1\_entropy = getEntropy(Sensor1\_DataSet);

**double** Sensor2\_entropy = getEntropy(Sensor2\_DataSet);

**double** Sensor3\_entropy = getEntropy(Sensor3\_DataSet);

System.***out***.println(Sensor1\_entropy+" "+Sensor2\_entropy+" "+Sensor3\_entropy);

**if**(Sensor1\_entropy>Sensor2\_entropy&&Sensor1\_entropy>Sensor3\_entropy)

**return** 1;

**else** **if**(Sensor2\_entropy>Sensor1\_entropy&&Sensor2\_entropy>Sensor3\_entropy)

**return** 2;

**else** **if**(Sensor3\_entropy>Sensor1\_entropy&&Sensor3\_entropy>Sensor2\_entropy)

**return** 3;

**else**

**return** 0;

}

**private** **double** getEntropy(**int**[] ar)

{

/\*

\* Step 1: Creating the Hash map and corresponding key-value pairs\*/

**int**[][] matrix = getHash(ar);

/\*

\* Step 2: Find probability of each entrySet\*/

**double**[] probAr = getProb(matrix);

**double**[] invProbAr = getInverseProb(probAr);

**double**[] logProbAr = getLogOfInvProb(invProbAr);

**double** entropy = 0;

**for**(**int** i = 0;i<probAr.length;i++)

entropy += probAr[i]\*logProbAr[i];

**return** entropy;

}

**private** **double**[] getLogOfInvProb(**double**[] ar)

{

**double**[] temp = **new** **double**[ar.length];

**for**(**int** i = 0;i<ar.length;i++)

{

temp[i] = Math.*log*(ar[i])/Math.*log*(2);

}

**return** temp;

}

**private** **double**[] getInverseProb(**double**[] ar)

{

**double**[] temp = **new** **double**[ar.length];

**for**(**int** i = 0;i<ar.length;i++)

{

temp[i] = 1/ar[i];

}

**return** temp;

}

**private** **double**[] getProb(**int**[][] matrix)

{

**double**[] tempProb = **new** **double**[matrix.length];

**double** totalMessage = 0;

**for**(**int** i = 0;i<matrix.length;i++)

totalMessage += matrix[i][1];

**for**(**int** i = 0;i<tempProb.length;i++)

tempProb[i] = matrix[i][1]/totalMessage;

**return** tempProb;

}

/\*

\* this method returns the matrix of key-value pair for each dataset

\* Here key is each of the data set entry

\* Here value is the frequency of the corresponding key\*/

**private** **int**[][] getHash(**int**[] ar)

{

HashMap<Integer, Integer> myHash = **new** HashMap<Integer, Integer>();

**for**(**int** i = 0;i<ar.length;i++)

{

**if**(myHash.containsKey(ar[i])==**true**)

myHash.replace(ar[i], myHash.get(ar[i])+1);

**else**

myHash.put(ar[i], 1);

}

**int**[][] matrix = **new** **int**[ar.length][2];

**int** matrixIndex = 0;

**for**(**int** key: myHash.keySet())

{

matrix[matrixIndex][0] = key;

matrix[matrixIndex][1] = myHash.get(key);

matrixIndex++;

}

**int**[][] finalMatrix = **new** **int**[matrixIndex][2];

**for**(**int** i = 0;i<matrixIndex;i++)

{

finalMatrix[i][0] = matrix[i][0];

finalMatrix[i][1] = matrix[i][1];

}

**return** finalMatrix;

}

**private** **int**[] getDataSet(String s)

{

String[] tempStrAr = s.split("-");

**int**[] tempIntAr = **new** **int**[tempStrAr.length];

**for**(**int** i = 0;i<tempIntAr.length;i++)

{

tempIntAr[i] = Integer.*parseInt*(tempStrAr[i]);

}

**return** tempIntAr;

}

}

**TABLE OF COMPONENETS**

|  |  |
| --- | --- |
| Name of Component | Cost Of component(Rs) |
| PIC Microcontroller 16f4520 | 250 |
| Independent Server | 1100 |
| Liquid Crystal Display 16x2 | 150 |
| MQ 135 | 250 |
| PCB | 30 |
| Connecting Wires | 250 |

Table No 1

**CONCLUSION**

This project report introduced **Redundancy Elimination Algorithm** written on the concept of entropy for the prioritization of sensors. We presented an innovative way to prioritize a particular sensor out of an sensor array for efficient memory allocation in microcontrollers.

Moreover it is an uncomplicated and simple task that greatly reduces the time, energy and labor required to do the same work. Here we have implemented a case where we are receiving data from multiple channels over a single hub from certain pool of networked sensors but the memory to store this data is available for only a few sensors.

It has greatly helped in reducing the efficiency of the Internet in the course of exchanging Internet data. It has also helped to remove duplicate strings from within arbitrary network flows, has emerged as a powerful technique to improve the efficiency of network links in the face of repeated data and has been widely used.

Redundancy elimination middle-boxes are being widely deployed to improve the effective bandwidth of network access links of enterprises and data centers alike, and for improving link loads in small ISP networks.

**REFERENCES USED**

1. Paul C. Shields, “The Entropy Theorem via Coding Bounds," IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 37, NO. 6, NOVEMBER 1991.
2. Rudolf Ahlswede; Ning Cai.; “Two Proofs of Pinsker’s Conjecture Concerning Arbitrarily Varying Channels," IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 37, NO. 6, NOVEMBER 1991.
3. Dimitris Nicolas Politis; " Moving Average Processes and Maximum Entropy," IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 38, NO. 3, MAY 1992
4. WEBSITES: www.electronicshub.com