**System Software design**

After designing the hardware now its time to design the software part of the project .This part include the software used for the developing the program for the system and description of the code developed

**Platform used for project (**Arduino **)**

Arduino is an open-source electronics platform based on easy-to-use hardware and software. [Arduino boards](https://www.arduino.cc/en/Main/Products) are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the [Arduino programming language](https://www.arduino.cc/en/Reference/HomePage) (based on [Wiring](http://wiring.org.co/)), and [the Arduino Software (IDE)](https://www.arduino.cc/en/Main/Software), based on [Processing](https://processing.org/).

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of [accessible knowledge](http://forum.arduino.cc/) that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The [software](https://www.arduino.cc/en/Main/Software), too, is open-source, and it is growing through the contributions of users worldwide.

**Program (code) editor (**Arduino IDE 1.8.13**)**

Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

**Uploading the code**

Before uploading your sketch, you need to select the correct items from the **Tools > Board** and **Tools > Port** menus. The [boards](https://www.arduino.cc/en/guide/environment#boards) are described below. On the Mac, the serial port is probably something like **/dev/tty.usbmodem241** (for an Uno or Mega2560 or Leonardo) or **/dev/tty.usbserial-1B1** (for a Duemilanove or earlier USB board), or **/dev/**tty.USA19QW1b1P1**.1** (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be **/dev/ttyACMx**, **/dev/ttyUSBx** or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the **Upload** item from the **Sketch** menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino **bootloader**, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

**Libraries**

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the **Sketch > Import Library** menu. This will insert one or more **#include** statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its **#include** statements from the top of your code.

There is a [list of libraries](https://www.arduino.cc/en/Reference/Libraries) in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch.

**Program design for system**

**Libraries used in program**

**Radio Head library(#include <RH\_ASK.h>)**

This is the RadioHead Packet Radio library for embedded microprocessors. It provides a complete object-oriented library for sending and receiving packetized messages via a variety of common data radios and other transports on a range of embedded microprocessors.

**SPI library(#include <SPI.h>)**

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices:

**OneWire library (<OneWire.h>)**

OneWire lets you access 1-wire devices made by Maxim/Dallas, such as temperature sensors and ibutton secure memory. For temperature sensors, the DallasTemperature library can be used with this library.

**DallasTemperature library (<DallasTemperature.h>)**

This library support the temperature sensor DS18B20 as dallastemperature is manufacturer of it

**Algorithm**

**Flowchart**

**Program-:**

**Program at transmitting terminal**#include <RH\_ASK.h> //include a radio head library for Rf module interface

#include <SPI.h> //Include dependent SPI Library for communicating with transmitter

#include <OneWire.h> //include a OneWire library for temperature sensor

#include <DallasTemperature.h>

#define ONE\_WIRE\_BUS 0 // Data wire is plugged into digital pin 0 on the Arduino

OneWire oneWire(ONE\_WIRE\_BUS); // Setup a oneWire instance to communicate with any OneWire device

DallasTemperature sensors(&oneWire); // Pass oneWire reference to DallasTemperature

RH\_ASK rf\_driver; // Create Amplitude Shift Keying Object

//\*\*\*\*\*\*\*FUNCTION INITIALIZATION\*\*\*\*\*\*\*\*

float read\_mq7(void); //initialize function to read carbon monoxide value

float read\_mq2(void); //initialize function to read smoke gas value

float read\_ph(void); //initialize function to read PH value of solution

float get\_temperature(void); //initialize function to read temperature value of solution

void setup()

{

Serial.begin(9600);

rf\_driver.init(); // Initialize ASK Object

}

void loop() {

float PH, Temperature, MQ7value,MQ2value; // initialize the variables to store the input value of the sensors

String transmit ,transmit\_msg; // Define output strings

PH = read\_ph(); //call the function to measure the get the PH data

transmit\_msg=String(PH); //convert the data into string for transmitting through the rf module

transmit = "PH value " + transmit\_msg; //concatenate the msg string and converted data to transmit

const char \*msg\_ph = transmit.c\_str(); // Compose output character

rf\_driver.send((uint8\_t \*)msg\_ph, strlen(msg\_ph)); //transmit the data using the in build function of RH library

rf\_driver.waitPacketSent();//wait for the data to be send

delay(200);

//same procedure for the temperature value

Temperature = get\_temperature();

transmit\_msg=String(Temperature);

transmit = "Temperature " + transmit\_msg;

const char \*msg\_temperature = transmit.c\_str();

rf\_driver.send((uint8\_t \*)msg\_temperature, strlen(msg\_temperature));

rf\_driver.waitPacketSent();

delay(200);

//same procedure for the carbon monoxide value

MQ7value = read\_mq7();

transmit\_msg=String(MQ7value);

transmit = "carbon monoxide " + transmit\_msg;

const char \*msg\_mq7 = transmit.c\_str();

rf\_driver.send((uint8\_t \*)msg\_mq7, strlen(msg\_mq7));

rf\_driver.waitPacketSent();

delay(200);

//same procedure for the smoke gas value

MQ2value = read\_mq2();

transmit\_msg = String(MQ2value);

transmit = "smoke gas " + transmit\_msg;

const char \*msg\_mq2 = transmit.c\_str();

rf\_driver.send((uint8\_t \*)msg\_mq2, strlen(msg\_mq2));

rf\_driver.waitPacketSent();

delay(200);

}

//\*\*\*\*\*\*\*DEFINING THE FUNCTION\*\*\*\*\*\*\*\*

//defining the function to read the ph value

float read\_ph(void)

{

const int analogInPin = A0; //variable to store the data read by analog pin 0

unsigned long int avgValue; // variable to store the average of sensor value taken

int buf[10],temp; // variable to store the 10 samples of ph measured and a ''temp'' to store temporary value

for(int i=0;i<10;i++) //loop to take a 10 samples of the The PH value

{

buf[i]=analogRead(analogInPin); //read the ph value

delay(10);

}for(int i=0;i<9;i++) //sort the analog values from small to large

{

for(int j=i+1;j<10;j++)

{

if(buf[i]>buf[j])

{

temp=buf[i];

buf[i]=buf[j];

buf[j]=temp;

}

}

}

avgValue=0;

for(int i=2;i<8;i++) //take the average value of 6 center sample

avgValue+=buf[i];

float pHVol=(float)avgValue\*5.0/1024/6; //convert the values in into volt

float phValue = -5.70 \* pHVol + 21.34; //convert the volt into pH value

return phValue; // return the PH value to function call.

delay(20);

}

//defining the function to read the temperature value

float get\_temperature(void)

{

float Celsius = 0; //define the variable to store the temperature value

sensors.begin(); // begin the sensors library

sensors.requestTemperatures(); //read the temperature

Celsius = sensors.getTempCByIndex(0); //store the temperature value in the variable

return Celsius; //return the temperature value to function call

}

//defining the function to read the carbon monoxide value

float read\_mq7 (void)

{

int pinbuzzer = 11; //assign the buzzer pin to digital 11 pin of Arduino

int threshold\_level=2; //assign the threshold pin of sensor to digital pin 2 of arduino

float pinSensor = A2; //variable to store the analog value of sensor from analog pin2 of arduino

int THRESHOLD = 300; //set the threshold level according to the requirement

pinMode(pinbuzzer, OUTPUT); //assign pin as output

pinMode(pinSensor, INPUT); //assign pin as input

pinMode(threshold\_level,INPUT); //assign pin as input

int analogValue = analogRead(pinSensor); //read the analog value of carbon monoxide

return analogValue ; //return the measured value

digitalWrite(pinbuzzer, LOW); //low the buzzer

if (analogValue >= THRESHOLD ) //if the read value is above the threshold

digitalWrite(pinbuzzer,HIGH ); //then sound the buzzer

else

digitalWrite(pinbuzzer,LOW ); //if not then don’t sound

}

//defining the function to read the smoke gas value

float read\_mq2 (void)

{

int pinbuzzer = 11; //assign the buzzer pin to digital 11 pin of arduino

int threshold\_level=1; //assign the threshold pin of sensor to digital pin 1 of arduino

float pinSensor = A1; //variable to store the analog value of sensor from analog pin2 of arduino

int THRESHOLD = 450; //set the threshold level according to the requirement

pinMode(pinbuzzer, OUTPUT); //assign pin as output

pinMode(pinSensor, INPUT); //assign pin as input

pinMode(threshold\_level,INPUT); //assign pin as input

int analogValue = analogRead(pinSensor); //read the analog value of smoke gas

return analogValue ; //return the measured value to function call

digitalWrite(pinbuzzer, LOW); //low the buzzer

if (analogValue >= THRESHOLD ) //if the read value is above the threshold

digitalWrite(pinbuzzer,HIGH ); // then sound the buzzer

else

digitalWrite(pinbuzzer,LOW ); //if not then don’t sound

}

**Program at receiving and display terminal**

#include <RH\_ASK.h> // Include dependent SPI Library

#include <SPI.h> // Create Amplitude Shift Keying Object

RH\_ASK rf\_driver;

#include <LiquidCrystal.h> //include the liquid crystal display library

LiquidCrystal lcd(7, 6, 5, 4, 3, 2); // Creates an LCD object. Parameters: (rs, enable, d4, d5, d6, d7)

String str\_out; //string to store the received data

void setup()

{

// Initialize ASK Object

rf\_driver.init();

// set up the LCD’s number of columns and rows:

lcd.begin(20, 4);

// Clears the LCD screen

lcd.clear();

}

void loop()

{

// Set buffer to size of expected message

uint8\_t buf[11];

uint8\_t buflen = sizeof(buf);

// Check if received packet is correct size

if (rf\_driver.recv(buf, &buflen))

{

str\_out = String((char\*)buf); //store the received msg in variable

// set the cursor to column 0, line 1

// (note: line 1 is the second row, since counting begins with 0):

lcd.setCursor(0, 1);

lcd.print(str\_out );

delay(3000);

}

else {

lcd.print("data not received" );

}

}