**National Institute Of Technology, Rourkela**



Project Report

ON

IoT enabled RADAR using Arduino Nani and Node MCU with

ZigBee interface

**By**

Gopala Krishna Abba

**Under the guidance of :**

S. K. Das

**Abstract**

This paper will perform as an instruction manual for designing an intrusion detection system (IDS) using a low cost IoT radar system that uses Arduino Nano and Node MCU for data processing and communication respectively and employs ZigBee interface to transmit data (radar data) to remote monitoring system. The project aims to showcase the feasibility of using IoT technology for radar applications and demonstrate monitoring capabilities.

**Introduction**

An Intrusion Detection System (IDS) is a monitoring system that detects suspicious activities and generates alerts when they are detected. Based upon these alerts, a security operations centre (SOC) analyst or incident responder can investigate the issue and take the appropriate actions to remediate the threat.

RADAR stands for Radio Detecting and Ranging and as indicated by the name, it is based on the use of radio waves. Radars send out electromagnetic waves similar to wireless computer networks and mobile phones. It is radiolocation system that uses radio waves to determine the distance, angle, and radial velocity of objects relative to the site. It is used to detect and track aircraft, ships, spacecraft, guided missiles, and motor vehicles, and map weather formations, and terrain. A radar system consists of a transmitter producing electromagnetic waves in the radio or microwaves domain, a transmitting antenna, a receiving antenna (often the same antenna is used for transmitting and receiving) and a receiver and processor to determine properties of the objects. Radio waves from the transmitter reflect off the objects and return to the receiver, giving information about the objects' locations and speeds.

When an object enters the line of transmission a signal is transmitted. The distance of the object from the radar is calculated and sent to the base station. the input of the radar is sent to the Arduino through USB serial port and FFT operation is done to get target power. This value is sent to ML model that can detect the material of the object using MATLAB.

PCB design, or Printed Circuit Board design, is a critical aspect of electronics engineering. It involves creating the physical layout of a PCB, which is the foundation for assembling electronic components and creating functional electronic devices. A PCB provides a platform for connecting and mounting electronic components. It offers a compact and organized way to interconnect these components, reducing wire clutter and enhancing reliability.

**Materials Used**

Hardware

* Arduino Nano
* Node MCU ESP 8266
* HB 100 Microwave Module
* XBEE Pro S2C Module
* XBEE Explorer Board
* Bread Board
* USB A to B Cable
* Sensor
* Amplifier circuit

Software

* MATLAB
* React
* Thingspeak

**PROCEDURE**

**HARDWARE**

**TEST CIRCUITS:**

**>>ZIGBEE/XBEE COMMUNICATION:**

**STEP1**- Program the XBEE-firmware using the XTCU software.

* **STEP1**: Download and install the XCTU software.

( LINK : <https://www.digi.com/resources/documentation/digidocs/90001526/tasks/t_download_and_install_xctu.htm> )

* **STEP2**: installing firmware to both the XBEE modules using XBEE programmer board

(NOTE: Individual programmer boards are needed for individual XBee )

PROCEDURE:

* Open XCTU software and click on “Discover boards”.
* Select the COM port to which the XBee module is connected and click on “Next”.
* Keep the default settings and click on “Finish”
* Now, on the pop-up window, click on “Add Selected Devices”.
* Now, the XBee module will appear on the left side of the window. Click on it to update the user interface.
* To update the firmware, click on “Update”, select “802.15.4 TH” in the Function set and select the newest firmware in the Firmware version and click on update. After this, a pop-up window will appear. Click on YES.
* Enter any 4 digits as PAN ID. PAN ID is a personal area network (PANs) identifier. Each network must be given a unique ID.

**NOTE**: Make sure that for both the XBee modules, the same PAN ID is entered. This indicates that both the XBee modules are in the same network.

* Give any 4 digits for the Destination Address. This same number must be entered as Source Address for the other XBee module.
* Give any 4 digits for the Source Address. This same number must be entered as the Destination Address for the other XBee module.
* Set one device as “Coordinate [1]” and the other device as “End Device [0]”.
* Now, click on Write in the top bar. After that is successful, the Symbol changes from “E” to “C” (End Device to Coordinator, as we have set. For the other XBee module, it should be “E”).

**Note:**As mentioned in the above steps, this must be carried out for both the XBee modules. The only difference between both will be that the Source and Destination Addresses are opposite for the End Device and Coordinator and one must be selected as the End Device and the other as “Coordinator”. In the “Coordinator Enable” drop-down list, make sure both the XBee modules are given the same PAN ID.

COM 5 is for nodeMCU (receiving side) and COM 14 is for Arduino-nano (Sending side).

**STEP2** - Connecting the XBEE’s with processors as shown in the connection diagram. (Fig-1 and Fig-2)

* **RECEIVER CIRCUIT:**

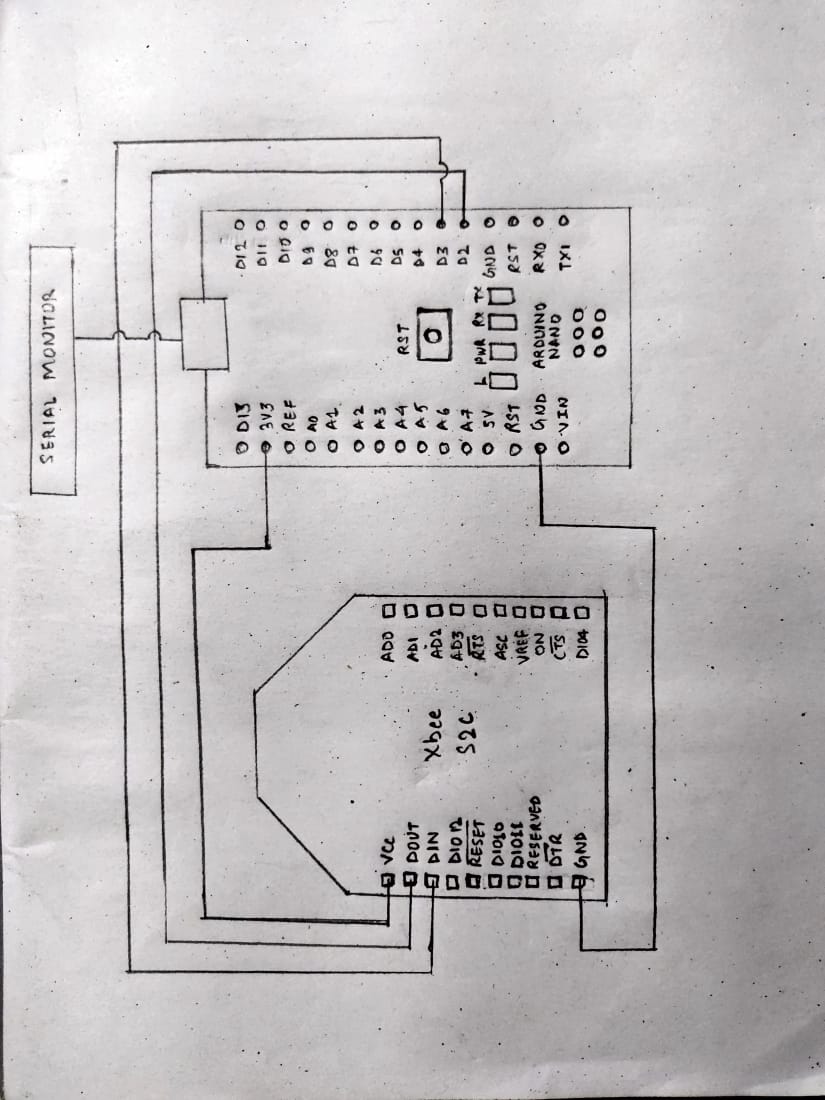


FIG-1

* **TRANSMITTER CIRCUIT:**

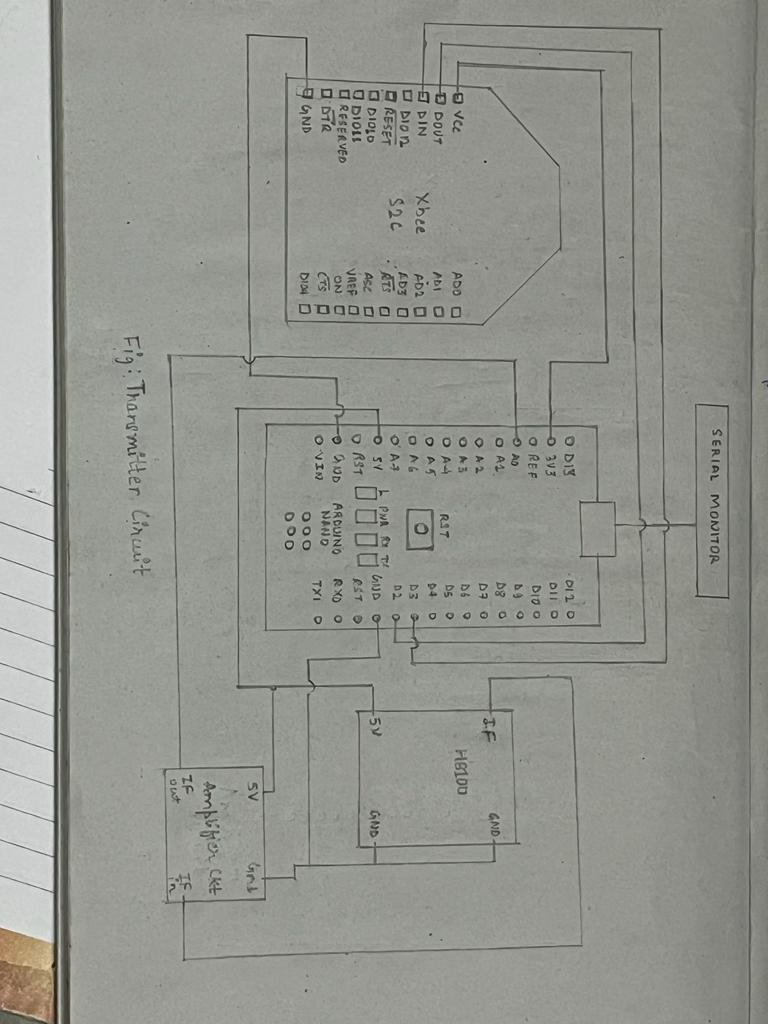


FIG-2

# STEP3 - Program NodeMCU on Arduino IDE

( link: <https://www.instructables.com/How-to-Program-NodeMCU-on-Arduino-IDE/> )

**STEP4** - Code the Arduino-nano and Node-MCU using the Arduino IDE.

( IDE LINK : <https://www.instructables.com/How-to-Program-NodeMCU-on-Arduino-IDE/> )

* **RECEIVER CODE for testing connection of Node-MCU/XBEE-END**

#include<SoftwareSerial.h>

int led = 10;

int received = 0;

int i;

//For communicating with zigbee

SoftwareSerial zigbee(13,12);

void setup()

{

Serial.begin(9600);

zigbee.begin(9600);

pinMode(led, OUTPUT);

}

void loop()

{

//check if the data is received

if (zigbee.available() > 0)

{

received = zigbee.read();

//if the data is 0, turn off the LED

if (received == '0')

{

Serial.println("Turning off LED");

digitalWrite(led, LOW);

}

//if the data is 1, turn on the LED

else if (received == '1')

{

Serial.println("Turning on LED");

digitalWrite(led, HIGH);

}

}

}

* **TRANSMITTER CODE for testing the connection of Arduino-NANO/XBEE-COORDINATOR**

#include "SoftwareSerial.h"

SoftwareSerial XBee(2,3);

boolean toggle = false;

void setup()

{

Serial.begin(9600);

XBee.begin(9600);

}

void loop()

{

//Toggle to 1 for initial output of XBEE

{

Serial.println("Turn on LED");

toggle = false;

XBee.write('1');

delay(1000); //1sec delay

}

//Toggle to 0 for next outputof XBEE

{

Serial.println("Turn off LED");

toggle = true;

XBee.write('0');

delay(1000); //1sec delay

}

}

**PRECAUTIONS:**

1. Make sure the connections are correct,
2. XBEE with the “END” tag is connected at the transmitter-side and

“COORDINATOR” tag is connected at the receiver-side.

1. Make sure the soldering is done properly. (For testing the soldered connection, multi-meter can be used.)

**OUTPUT OF THE ZIGBEE CONNECTION:**

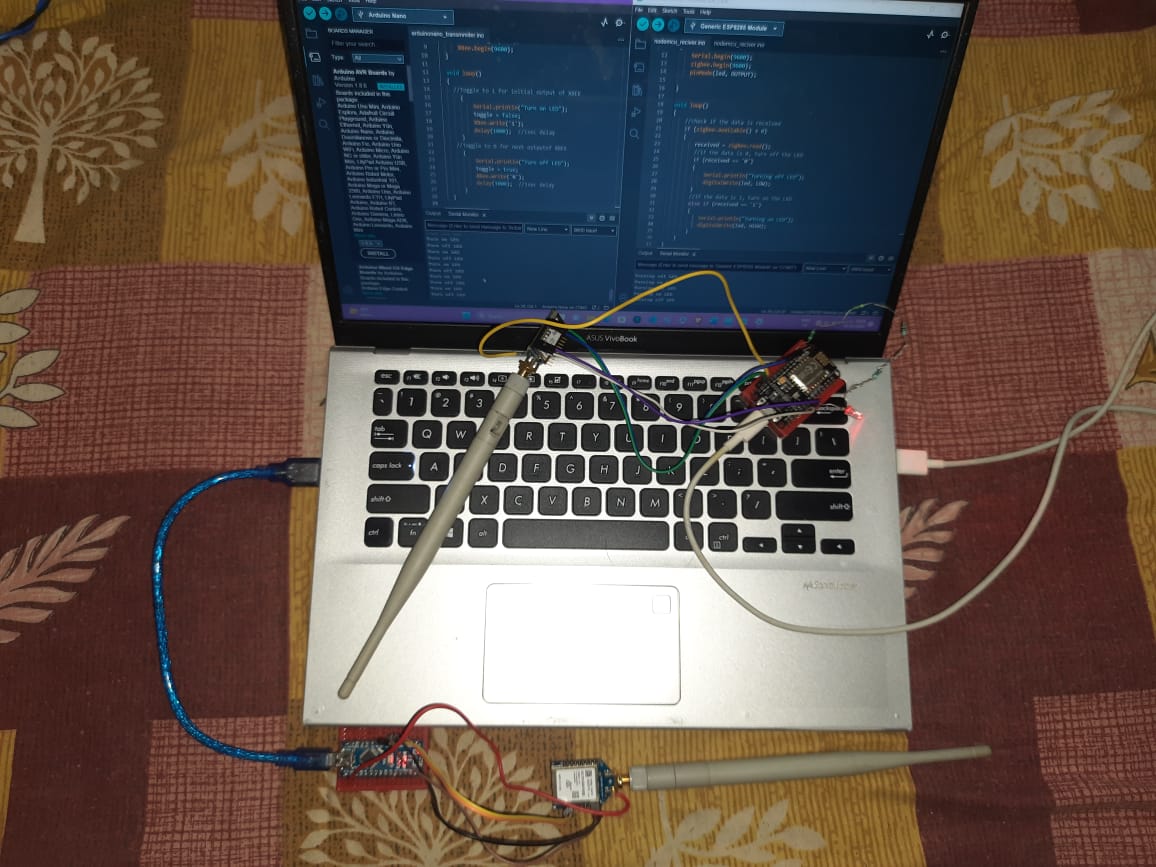


FIG-3

**>> Interfacing HB100 to Arduino-nano:**

**STEP1** - Connect HB100 and Arduino-nano according to the following circuit diagram.(Fig-

4)

* **CIRCUIT:**

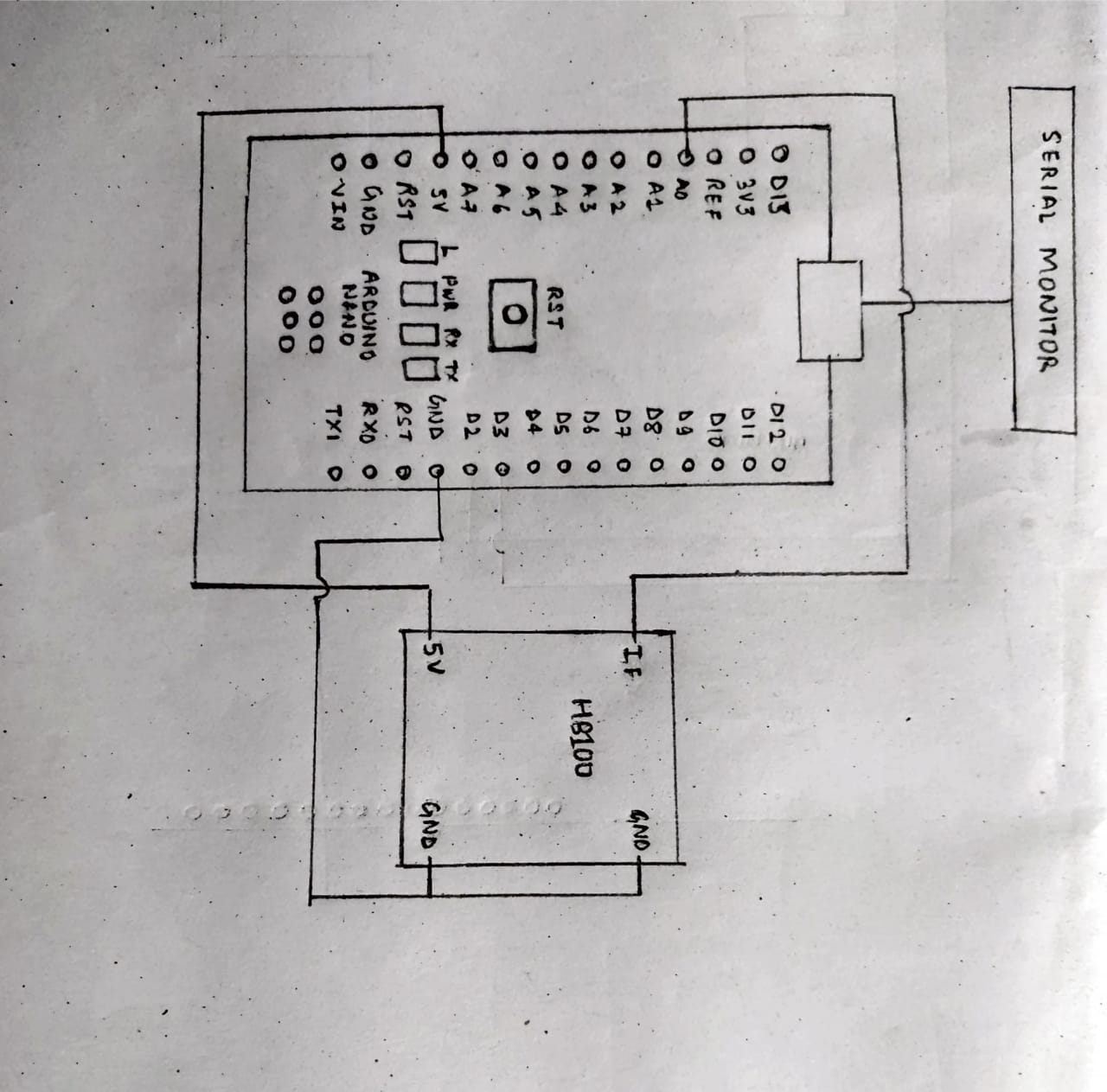


FIG-4

**STEP2 -** Code the Arduino-nano to read the input signal.

* **Arduino-nano Code:**

#include “SoftwareSerial.h”

void setup()

{

Serial.begin(9600);

}

void loop()

{

int c=analogRead(A0);

Serial.println(c);

delay(10);

}

**PRECAUTIONS:**

1. The HB100 should be oriented correctly.
2. Motion in circuit element will introduce noise.
3. Check the connections properly.

**OUTPUT:**

****

FIG-5:Connection of HB100 with Arduino-nano

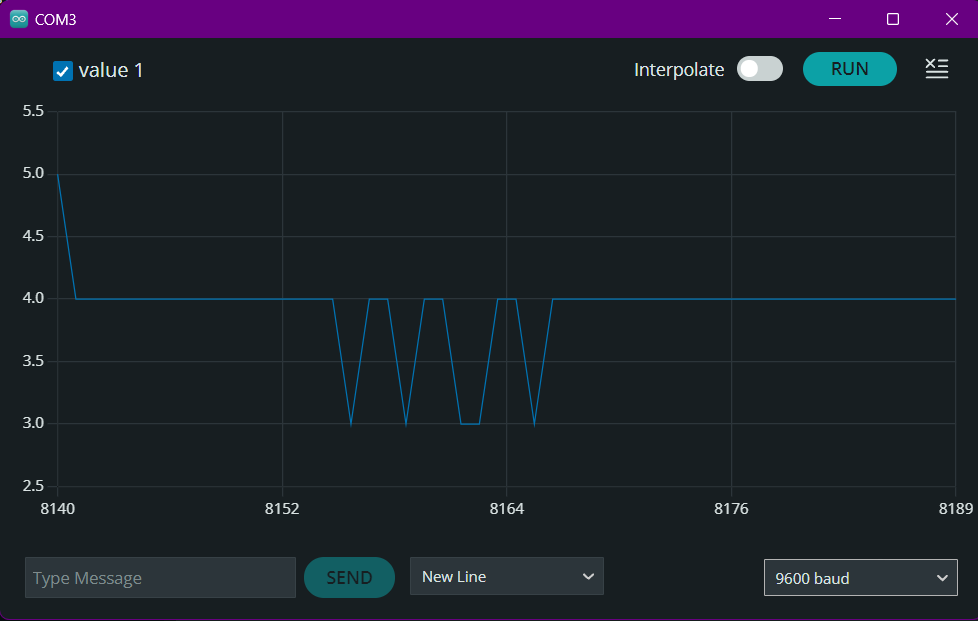


FIG-6: Output graph of HB100

In the hardware, we have the transmitter side that has Arduino nano, Xbee module and HB100 and the receiver side has the HB100 and NodeMCU. After initial connections, it was noticed that the signal strength was weak. Henceforth, implementation of an amplifier circuit was done as show below. Getting the necessary components and making connection gave us a stronger signal

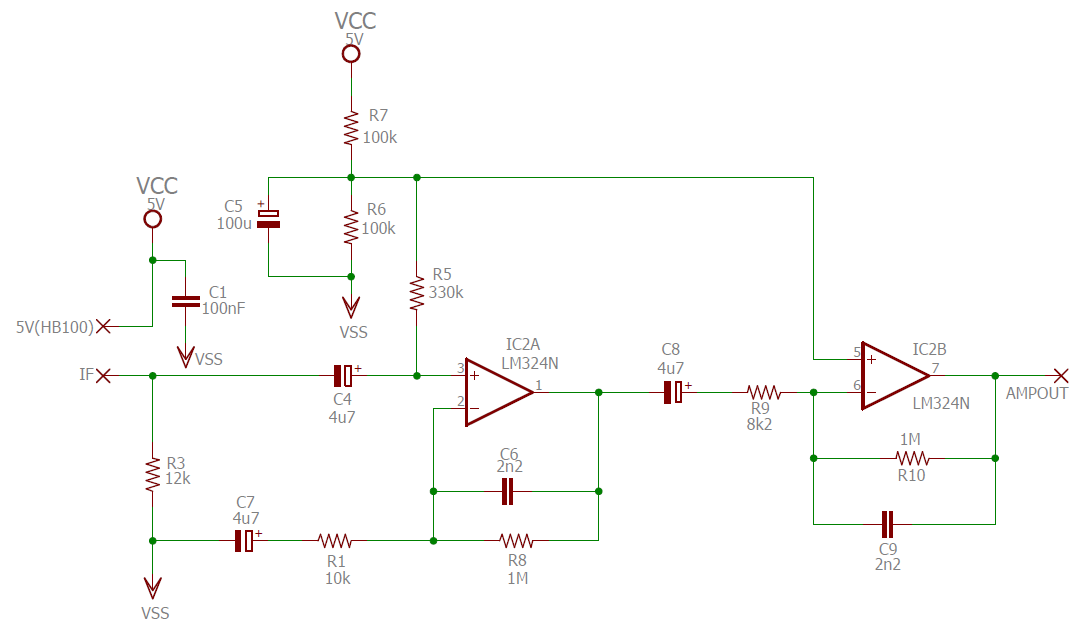


FIG-7: Amplifier Circuit

STEP1: Fast fourier transform (FFT) needs to be performed on the signal detection to separate out the clutter and doppler signal. From theory, the clutter will be at low frequency around 0Hz and Doppler will be at higher frequency. High pass filtering and noise thresholding is to be implemented and we can get our required target signal.

The MATLAB code is as follows

clc;

clear device;

close all;

% Specify the Excel file name and sheet name

filename = 'ML\_input.xlsx';

sheet = 'Sheet1';

% Create headers for the table

headers = {'Material', 'Frequency', 'Amplitude'};

% Open the Excel file for writing (create it if it doesn't exist)

%xlswrite(filename, headers, sheet, 'A1:C1');

% Initialize a row counter for adding data to the table

row = 2;

device = serialport("com3", 9600);

H = zeros(1000);

t = zeros(1000);

for c=1:1000

data = readline(device); %if your device uses serial print

H(c)=data;

t(c)=c;

end

fr=abs(str2num(readline(device)))

%s=spectrogram(H(1));

fre=ceil(fr);

subplot(2,1,1)

plot(t,H);

y=fft(H);

m = abs(y);

f = (0:length(y)-1)\*100/length(y);

%s=spectrogram(f)

subplot(2,1,2)

plot(f,m);

amp=m(fre)

% Simulate real-time data processing (replace this with your real data source)

%while true

% Simulate data acquisition (replace this with your data acquisition code)

material = 'Aluminium'; % Replace with your actual material data

frequency = fr; % Replace with your actual frequency data

amplitude = amp; % Replace with your actual amplitude data

% Create a row of data

rowData = {material, frequency, amplitude};

% Read the data from the Excel sheet as a table

data = readtable(filename, 'Sheet', sheet);

% Specify the column name for which you want to find the last row number

%column\_name = 'Material'; % Replace with the actual column name

% Find the last row number with data in the specified column

%last\_row = find(~isnan(data.(column\_name)), 1, 'last')

%last\_row()

last\_row = size(data, 1)

row=last\_row+2

pos=strcat('A',num2str(row))

% Append the new data to the Excel file

if(~(isnan(amp)))

xlswrite(filename, rowData, sheet, [pos]);

end

% Increment the row counter

%row = row + 1;

% You can introduce a pause to control the data acquisition rate

pause(1); % Adjust the time interval as needed

%end

clear device1;

device1 = serialport("com5", 9600);

pause(30);

write(device1,3,"int8");

data = readline(device1)

pause(30);

write(device1,1,"int8");

data = readline(device1)

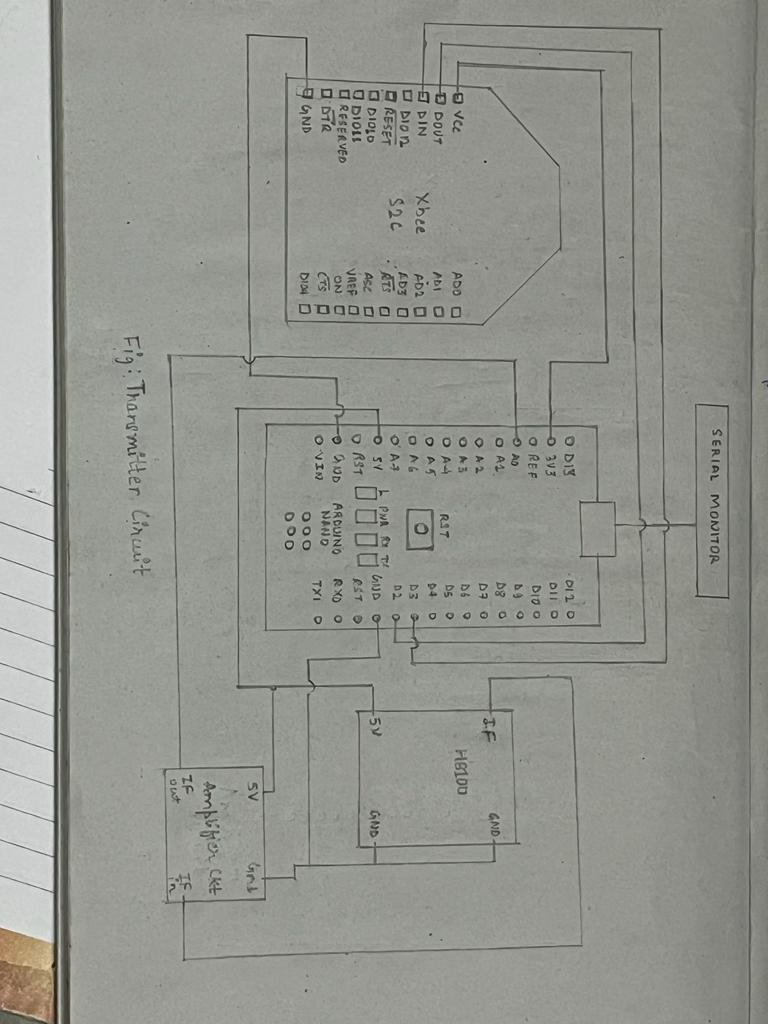
pause(20);

**PROTOTYPE:**

**STEP1:** Make the final circuit as shown in the figures (fig.8 and fig.9) as the test circuit.

* Fix the circuit to prevent noise and proper wireless communication.

**CIRCUIT:**



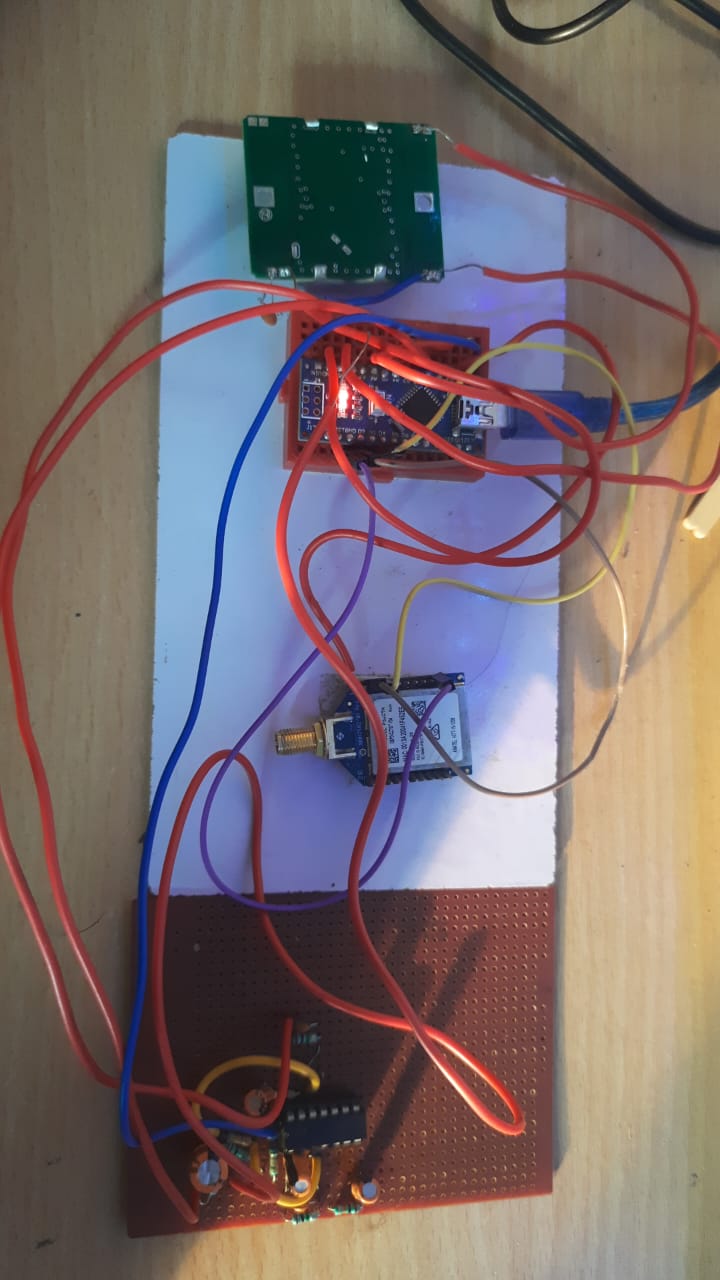


FIG-8: TRANSMITTER SIDE CIRCUIT

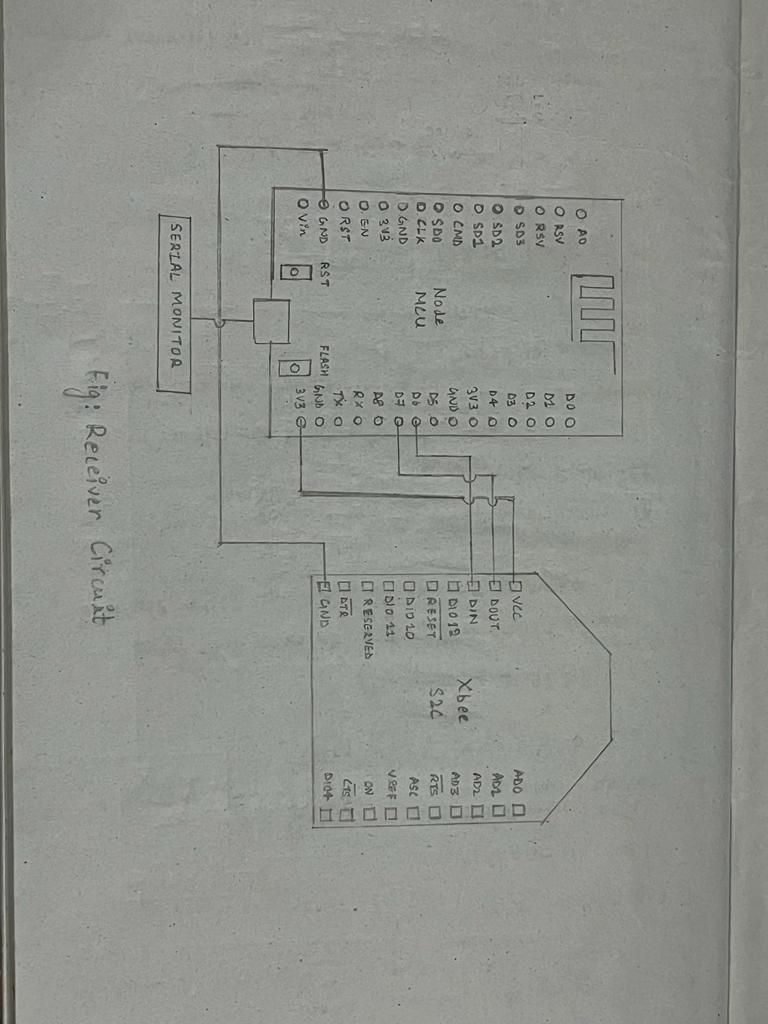


FIG-9 : RECEIVER SIDE CIRCUIT

**STEP2:**  Code Arduino-nano and Node MCU.

* **RECEIVER CODE:(NODE-MCU)**

#include <SoftwareSerial.h>

//Use this file to store all of the private credentials

//and connection details

int led=BUILTIN\_LED;

#define SECRET\_SSID "ASHU 9158" // replace MySSID with your WiFi network name

#define SECRET\_PASS "747J0(q8" // replace MyPassword with your WiFi password

#define SECRET\_CH\_ID 2275453 // replace 0000000 with your channel number

#define SECRET\_WRITE\_APIKEY "CEJ00W8KMQJ3SARE" // replace XYZ with your channel write API Key

#include "ThingSpeak.h"

#include <ESP8266WiFi.h>

char ssid[] = SECRET\_SSID; // your network SSID (name)

char pass[] = SECRET\_PASS; // your network password

int keyIndex = 0; // your network key Index number (needed only for WEP)

WiFiClient client;

unsigned long myChannelNumber = SECRET\_CH\_ID;

const char \* myWriteAPIKey = SECRET\_WRITE\_APIKEY;

// Initialize our values

//int number1 = 0;

// int number1 = random(0,100);

// String myStatus = "";

SoftwareSerial zigBee(12,13);

void setup()

{

Serial.begin(9600);

zigBee.begin(9600);

WiFi.mode(WIFI\_STA);

ThingSpeak.begin(client); // Initialize ThingSpeak

pinMode(led,OUTPUT);

}

double p=0,y=0,t=0;

void loop()

{

// if(zigBee.available()>0){

// for(int i=0;i<1000;i++){

// if(zigBee.available()>0)

// {

// //if(i>3)

// {

// p=zigBee.read();

// Serial.println(p);

// }

// }

// //ThingSpeak.setField(1, p);

// }

digitalWrite(led,HIGH);

int incomingByte=0;

if (Serial.available() > 0) {

// read the incoming byte:

incomingByte = Serial.read();

// ThingSpeak.setField(1, incomingByte);

delay(1000);

digitalWrite(led,LOW);

}

// // int incomingByte=0;

// // if (Serial.available() > 0) {

// // // read the incoming byte:

// // incomingByte = Serial.read();

// // // ThingSpeak.setField(1, incomingByte);

// // // int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

// // // if(x == 200){

// // // Serial.println("Channel update successful.");

// // // }

// // // else{

// // // Serial.println("Problem updating channel. HTTP error code " + String(x));

// // // }

// // digitalWrite(led,LOW);

// // }

// Connect or reconnect to WiFi

if(WiFi.status() != WL\_CONNECTED){

Serial.print("Attempting to connect to SSID: ");

Serial.println(SECRET\_SSID);

while(WiFi.status() != WL\_CONNECTED){

WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network. Change this line if using open or WEP network

Serial.print(".");

delay(5000);

}

Serial.println("\nConnected.");

}

// digitalWrite(led,HIGH);

if(incomingByte!=0)

{

ThingSpeak.setField(1, incomingByte);

int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

if(x == 200){

Serial.println("Channel update successful.");

}

else{

Serial.println("Problem updating channel. HTTP error code " + String(x));

}

}

// set the fields with the values

// send data only when you receive data:

// if (Serial.available() > 0) {

// // read the incoming byte:

// int incomingByte = Serial.read();

// //ThingSpeak.setField(1, incomingByte);

// }

// ThingSpeak.setField(1, incomingByte);

// figure out the status message

// if((number1 > number2) && (number1 > number3)){

// myStatus = String("field1 is set as status");

// }

// else if((number2 > number1) && (number2 > number3)){

// myStatus = String("field2 is set as status");

// }

// else{

// myStatus = String("field3 is set as status");

// }

// // set the status

// ThingSpeak.setStatus(myStatus);

// write to the ThingSpeak channel

// int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

// if(x == 200){

// Serial.println("Channel update successful.");

// }

// else{

// Serial.println("Problem updating channel. HTTP error code " + String(x));

// }

digitalWrite(led,HIGH)

}

//}

* **TRANSMITTER CODE:(ARDUINO-NANO)**

#include "FreqPeriodCounter.h"

#include <SoftwareSerial.h>

SoftwareSerial XBee(2,3);

const byte counterPin = A0;

const byte counterInterrupt = A1; // = d3

FreqPeriodCounter counter(counterPin, micros);

void setup(void)

{ attachInterrupt(counterInterrupt, counterISR, CHANGE);

Serial.begin(9600);

pinMode(A0, INPUT);

XBee.begin(9600);

}

void loop(void)

{

int c=analogRead(A0);

//Serial.println(c);

if(c==0)

{

for(int i=0;i<1000;i++)

{

c=analogRead(A0);

Serial.println(c);

XBee.write(c);

}

digitalWrite(A1,LOW);

int period;

if(counter.ready()) period = counter.period;

Serial.println((float)1/period);

c=abs((float)1/period);

XBee.write(c);

digitalWrite(A1,HIGH);

//delay(1000);

}

}

void counterISR()

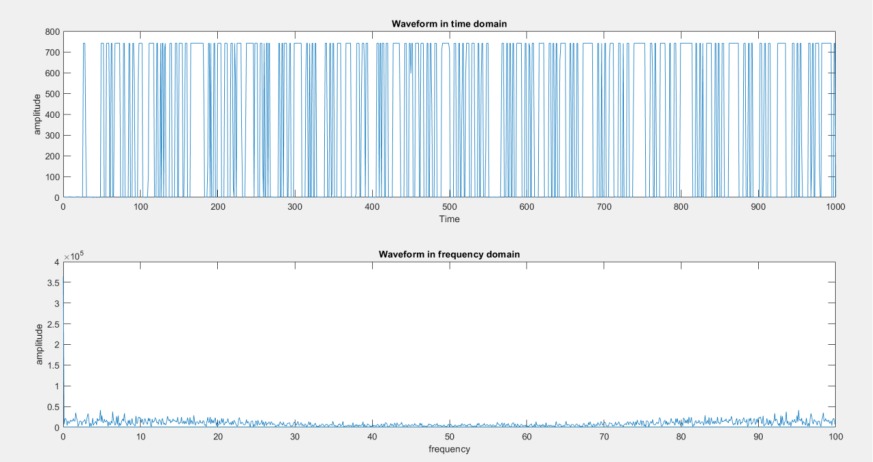
{ counter.poll();

}

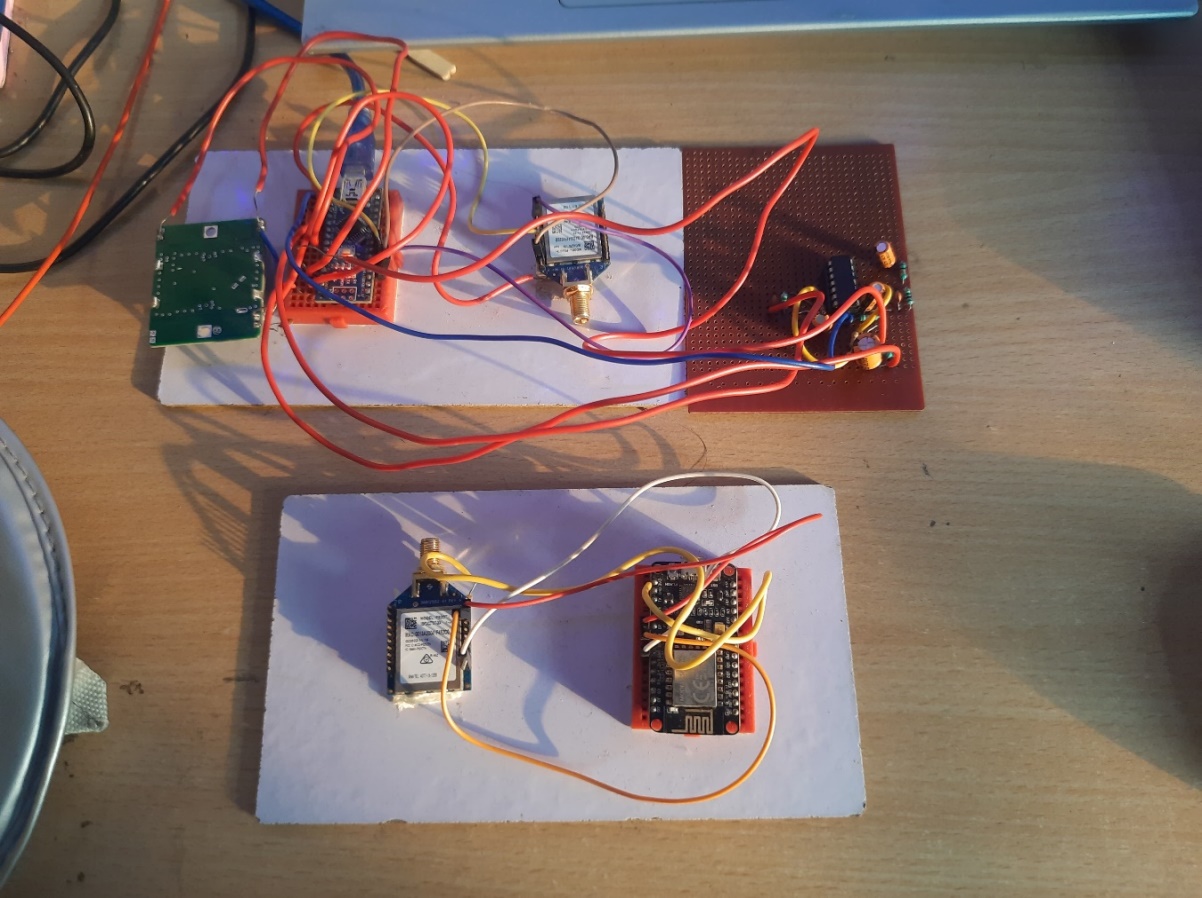
**PRECAUTION:**

1. The logic depends on relative peaks which might be generated due to wind.
2. HB100 is a dopler based RADAR sensor as a result direct distance calculation is not possible. (HB100 is a motion sensor)

**OUTPUT:**

****

**Fig 10 : FFT waveforms of MATLAB**



**Fig 11: Complete Hardware Setup**

**SOFTWARE**

The front end is a React application that displays the material of the detected object. This works in collaboration with ThingSpeak an open source IoT platform. ThingSpeak is an open-source Internet of Things (IoT) platform that allows users to collect, analyze, and visualize data from various sensors and devices. Users can send data to ThingSpeak channels using devices like Arduino, Raspberry Pi, or any sensor-equipped hardware. Once the data is on ThingSpeak, users can perform real-time analytics, create visualizations such as charts and graphs, and set up alerts based on specific conditions. For our project, the type of material detected can be stored. ThingSpeak's simplicity and versatility make it a popular choice for IoT projects, enabling seamless monitoring and control of connected devices and systems.

**Setting up Thingspeak**

1. Go to ThingSpeak official website ([IoT Analytics - ThingSpeak Internet of Things](https://thingspeak.com/))

2. Create an account

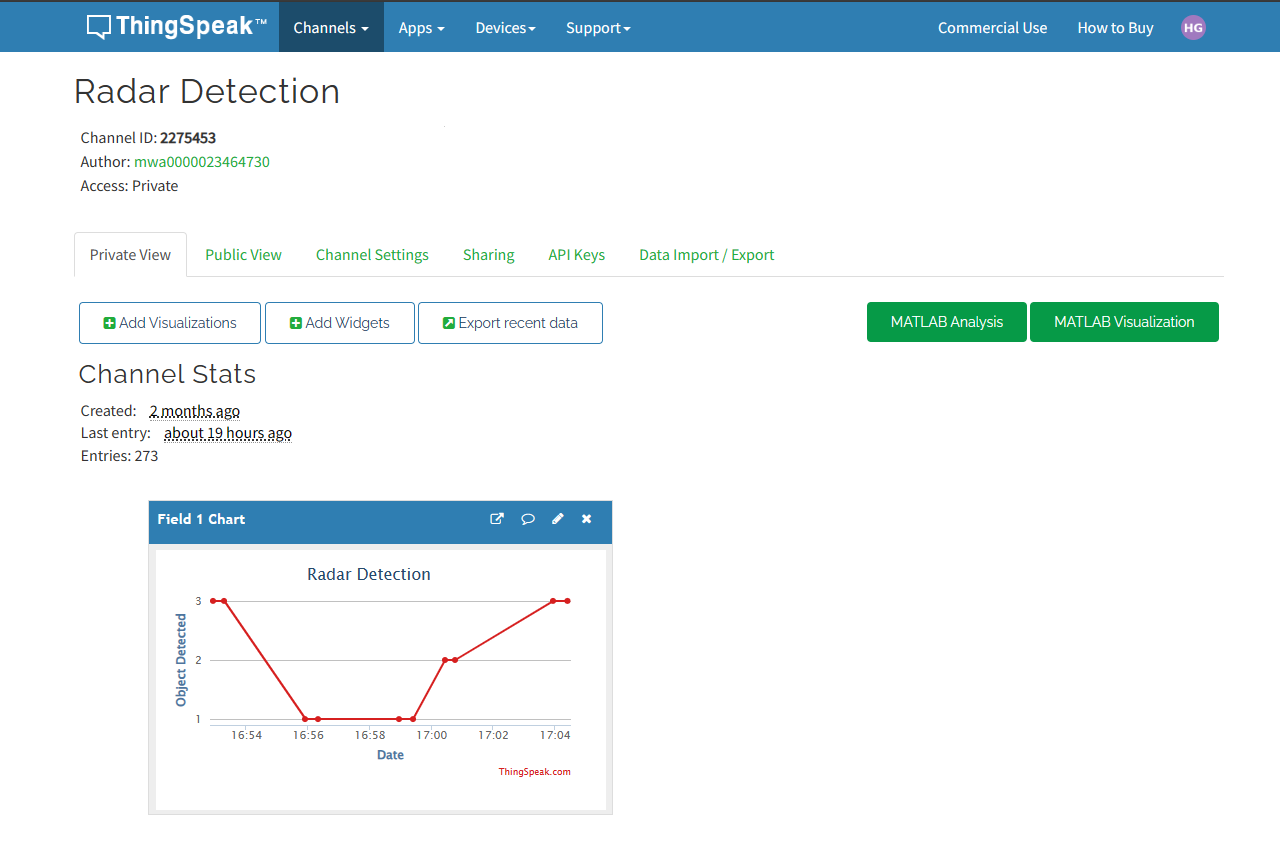
3. Create a channel. The name provided was Radar Detection

4. Add one field for the designated number that would be received from NodeMCU

5. Once the channel is created we note down the channel number, API keys and code to upload to NodeMCU and read field data for displaying on frontend.

6. On the main view, addition of visualization for the field data is done for better understanding.

The ThingSpeak main view along with visualization is as shown below



**Development of frontend**

* Pre-requisites

1. **Install Node.js and NPM** on your desktop. You can download Node at [Node.js (nodejs.org)](https://nodejs.org/en). Make sure to have **at least Node 14.0.0 version** as Create React App runs on Node 14 and its above version. For more assistance one can refer to [Downloading and installing Node.js and npm | npm Docs (npmjs.com)](https://docs.npmjs.com/downloading-and-installing-node-js-and-npm)
2. If Node is already installed, then **upgrade it to the latest version**. You may use the latest version of Node.js - NodeJS 21.
3. **Upgrade your NPM**. Create React App requires at least version 5.2 of NPM.
4. One needs to have a good editor to work with the project files. You can install, use, and create a React app in **Visual Studio Code**. Installation can be done at [Download Visual Studio Code - Mac, Linux, Windows](https://code.visualstudio.com/Download)

* Creating the react application

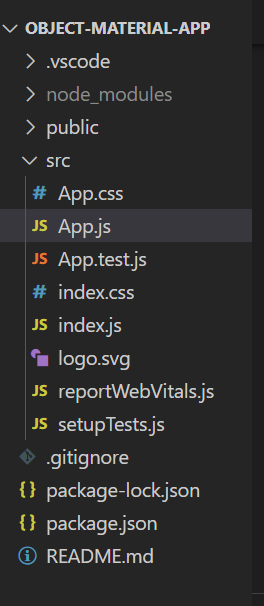
1. Open a new file in Visual Studio and in the terminal below type the following commands

npm install -g create-react-app

**npm create-react-app [name of app]**

for example: npm create-react-app object-material-app

1. After the installation of the required project files, your project structure may look something like this:



1. Modify the code for App.js as below

import React, { useState, useEffect } from 'react';

import './App.css';

function App() {

  const [material, setMaterial] = useState('');

  // Function to map designated numbers to materials

  const mapNumberToWord = (number) => {

    const numberMap = {

      1: 'Aluminium',

      2: 'Plastic',

      3: 'Wood'

      // Add more mappings as needed

    };

    return numberMap[number] || 'Unknown Material';

  };

  useEffect(() => {

    const fetchData = async () => {

      try {

        const response = await fetch(

          'https://api.thingspeak.com/channels/2275453/feeds.json?api\_key=H30GLTR4GGUUJ2GO&results=2'

        ); // This link is obtained from Read a Channel Field in API keys section of our channel on ThingSpeak

        const data = await response.json();

        if (data.feeds.length > 0) {

          // Map the received number to a word using the mapNumberToWord function

          const materialWord = mapNumberToWord(data.feeds[0].field1);

          setMaterial(materialWord);

        }

      } catch (error) {

        console.error('Error fetching data:', error);

      }

    };

    // Fetch data initially

    fetchData();

    // To update every 1 second

    const interval = setInterval(() => {

      fetchData();

    }, 1000);

    // Clear interval on component unmount

    return () => {

      clearInterval(interval);

    };

  }, []); // Empty dependency array ensures useEffect runs only once

  return (

    <div className="App">

      <h1>Object Material Detected Display</h1>

      <div>

        <p>Material: {material}</p>

      </div>

    </div>

  );

}

export default App;

1. Modify the App.css code as below

body {

  background-color: #353333;

  margin: 0;

  padding: 0;

  font-family: Arial, sans-serif;

  display: flex;

  justify-content: center;

  align-items: center;

  min-height: 100vh;

}

.App {

  display: flex;

  flex-direction: column;

  align-items: flex-start;

  padding: 20px;

  max-width: 600px;

  background-color: #474343;

  border-radius: 10px;

  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

}

h1 {

  font-size: 28px;

  margin-bottom: 20px;

  font-weight: bold;

  color: #fff;

}

p {

  font-size: 20px;

  margin-bottom: 20px;

  font-weight: bold;

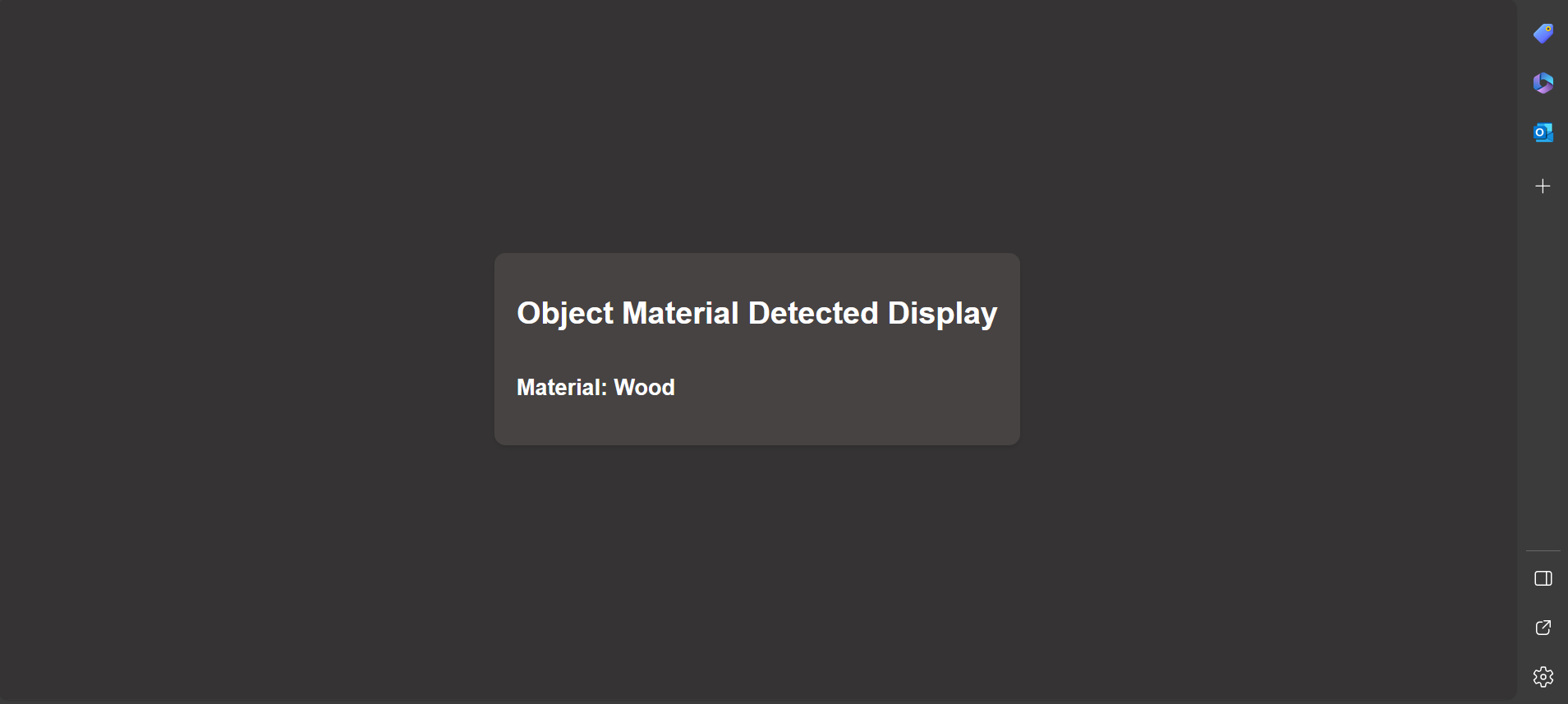
  color: #fff;

}

1. To run the app, in the terminal type the command

**npm run start**

The frontend is completed and will be as displayed below



Based on the object that enters the range of our sensor the designated number for that item is sent to ThingSpeak for visualization and with the usage of the API keys and fetch function, display of the same is done on the React frontend.

PCB Design Process

Schematic Design:

* Create a visual representation of the electronic circuit using EDA software, including component symbols and wire connections.
* Annotations and netlists help identify and cross-reference components.

Component Selection:

* Choose electronic components based on their datasheets, ensuring they meet project requirements.
* Select appropriate component footprints and maintain a component library for organization.

PCB Layout:

* Arrange components on the PCB, considering signal paths, thermal management, and mechanical constraints.
* Create copper traces to connect components, implement ground and power planes, and determine layer stackup.

Design Validation:

* Simulate the circuit to verify functionality and performance.
* Perform design rule checking (DRC), signal integrity analysis, and thermal analysis to address potential issues.

Gerber Files:

* Generate Gerber files that describe the PCB's design, including copper traces, component placements, and other details.
* Prepare documentation, including assembly drawings and a bill of materials (BOM), for the manufacturer.

**Result and discussion:**

In our project, the radar which is used is a motion sensor, works on the principle of doppler effect. Doppler effect is a work of frequency differences.

The output which we plot in a serial plotter, indicates different patterns (different wave pulse width and amplitude) for different kinds of objects intruded in the target region.

Though the radar, sensor fusion is a game changer in a perimeter security, radar being able to work in any weather conditions, has disadvantage too, as it cannot detect minute objects. Whereas for Xbee, as it is short ranged, to make use of it for higher range in real world, we need a radar with better specifications, and LoRa can be used for higher range.

The NodeMCU receives the detected material in serial port and sends a designated number to the ThingSpeak platform. On ThingSpeak, a channel for Radar Detection is created with one field for receiving the number. Observation of the graph can be done using visualizations on ThingSpeak platform. On the frontend created with React, the designated number is mapped to its' material/ object and is displayed accordingly..

**Conclusion:**

The implementation of the hardware portion for the project is done using Arduino Nano and NodeMCU was done for detection of the signal and know the material of the object. The display of the same is available as graph/visualization on ThingSpeak and on our React frontend. The implementation of the Gerber file is done and has been sent for fabrication thus completing the project.

**Precaution:**

* Make sure the connections are correct,
* XBEE with the “END” tag is connected at the transmitter-side and
* “COORDINATOR” tag is connected at the receiver-side.
* Make sure the soldering is done properly. (For testing the soldered connection, multi-meter can be used.)
* The HB100 should be oriented correctly.
* Motion in circuit element will introduce noise.
* Check the connections properly.
* The logic depends on relative peaks which might be generated due to wind.
* HB100 is a doppler based RADAR sensor as a result direct distance calculation is not possible. (HB100 is a motion sensor)

**REFERENCES AND LINKS:**

IEEE PAPERS-

* Aloysius Adya Pramudita,” Contactless Hand Gesture Sensor based on Array of CW Radar for Human to Machine Interface”, Member, IEEE, Lukas, Member, IEEE, Edwar,2017, DOI 10.1109/JSEN.2021.3073263

LINKS

* Download and install the XCTU software.

( LINK : <https://www.digi.com/resources/documentation/digidocs/90001526/tasks/t_download_and_install_xctu.htm> )

# Program NodeMCU on Arduino IDE

( link: <https://www.instructables.com/How-to-Program-NodeMCU-on-Arduino-IDE/> )

* Code the Arduino-nano and Node-MCU using the Arduino IDE.

( IDE LINK : <https://www.instructables.com/How-to-Program-NodeMCU-on-Arduino-IDE/> )

* Dataset to train the model

Link: https://www.sciencedirect.com/science/article/pii/S2352340922001330