purpose

Building a device for operating a remote radio station based on the Icom IC-706 model .

Building an affordable remote system using ESP32 AudioKit commercial development kits

Configuration

1 IC-706 radio / remote cable (OPC-581) / ESP32 AudioKit 2EA / Raspberry pi 3B or 4

Principle

of Operation* Reference Site:

1. https://lcbsweden.com/remote/part_icom_706/index.htm

2. https://www.oz9aec.net/ic-706

The remote program using Raspberry Pi was replaced using A1S.

Produce programs that are easy for anyone to use using Arduino IDE.

The A1S configured the Server and Client using the UDP protocol , and Audio uses the AudioKit library to reduce data volume and control data load using the Audio Compression Codec (OPUS).

*Use Library

1. https://github.com/pschatzmann/arduino-audiokit

2. https://github.com/sh123/esp32_opus_arduino

The rest use the Arduino primary library.

The IC-706 panel and the RIG body are connected and operated by serial data (TTL).

I have referred to Keepalive (how to maintain connection and separate data) that I can refer to on the reference site . The rest of the part was implemented by directly reading the cereal and interpreting it .

The panel sends the data related to the operation to the main body and the main body sends the data comprehensively again to show the received data on the panel . (The panel sends data that only changes the frequency, but the main body sends the entire LCD screen data including the frequency at once .)

Arduino has 3-4 processes running at the same time .

1. Serial Bridge Process (common)
2. Audio transmission/reception process (common)
3. Burton transmitting/receiving process (common)
4. Connection maintenance process (server only)
5. WiFi Management Process (Client only)

Four are running on Arduino . Arduino is basically not supported by multitasking, so it is implemented by multitasking as a Vtask implementation of ESP32, and four are running at the same time .

Servers and clients are exchanging audio and serial data using UDP protocols.

If you send or receive excessive data at once, the process will fail to process, causing you to slow down. There was no way to secure audio quality. (Improvements)

The power on/off process has been implemented for eye trickery.

Pressing the panel power button on the client transfers data to the server and the server attaches the PWK and GND of the league body in relay to turn on the power . The power off is turned off when the panel is pressed for more than 0.5 seconds, and the external 8V power supplied to the panel is turned off and on for about 1 second. At this time, the panel power appears to be off, but the 8V power is actually still being supplied to the panel . (This is also an improvement) If the button data is still being detected, the process load rate will increase, causing the A1S to malfunction.

The audio transmitting/receiving part uses the OPUS codec to compress and transmit data. It is transmitted in CD-class sound quality to improve quality, and the data load is significant. When lowered, the sound quality can make the damage or control more agile.

The PTT implementation part actually operates PTT as serial data rather than GND contact signal for the panel and body. The microphone connector has a PTT port, but internally, it is all operated by serial communication.

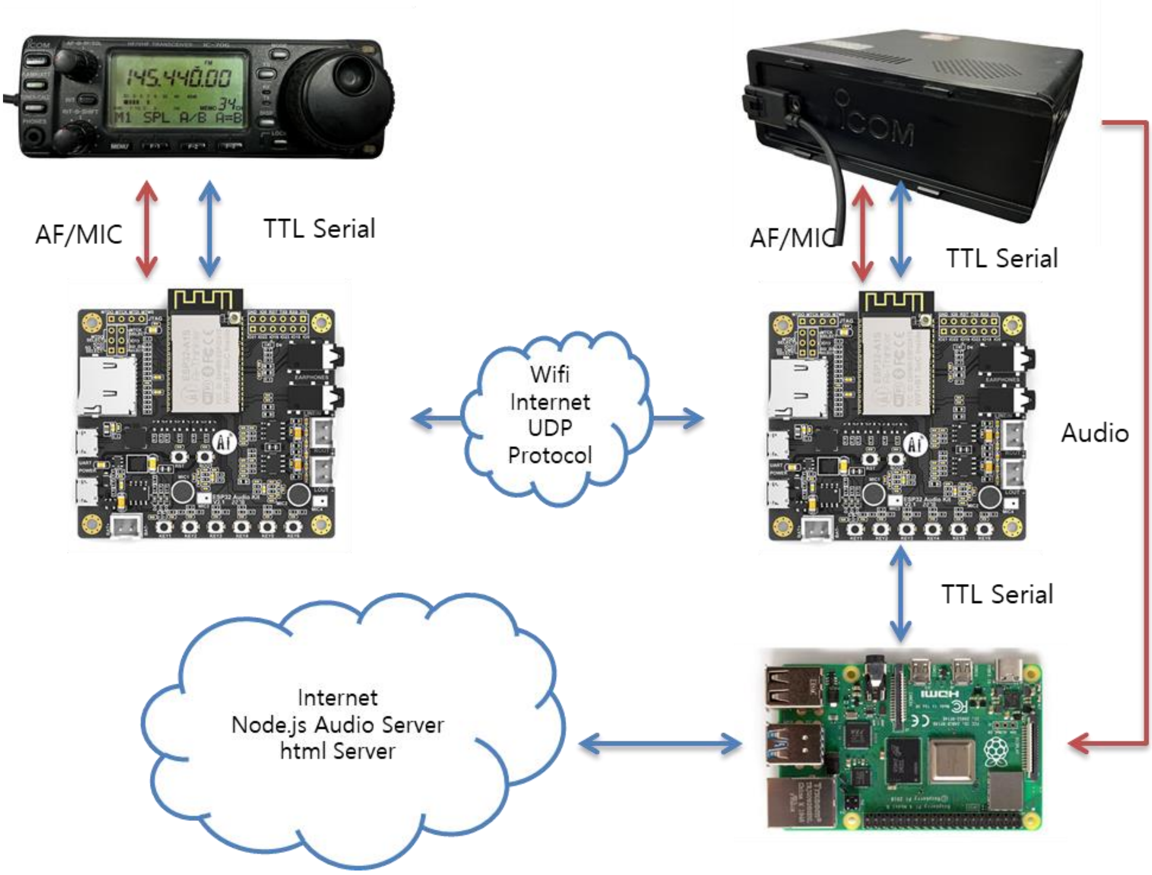


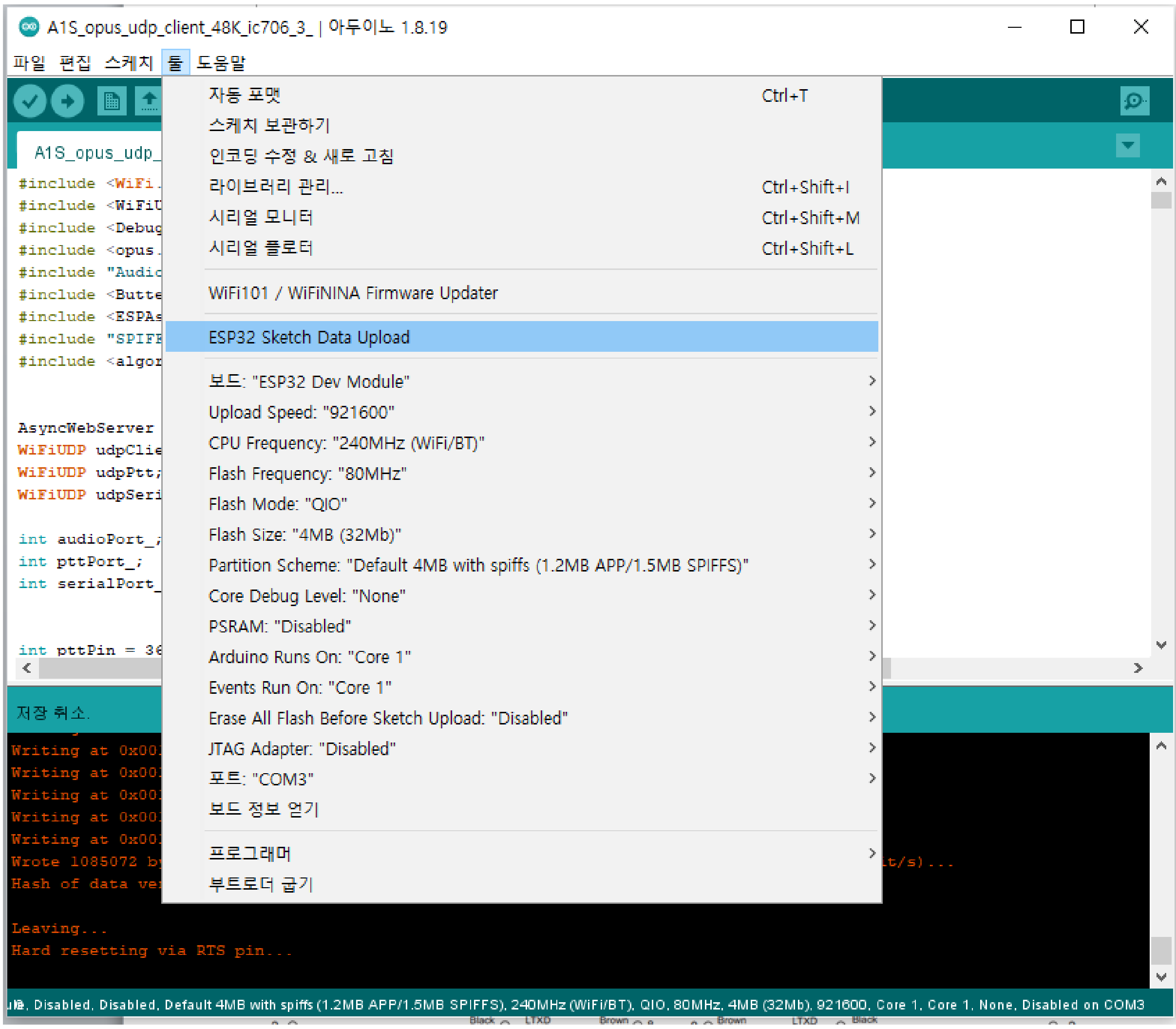
Figure 1 device configuration diagram

Use LRXD, LTXD, AF, MIC, PWK, 8V, GND on the OPC-581 cable to connect to ESP32 AudioKit A1S (hereinafter A1S).

Link :

Arduino Description

Client



The client has a WIFI Management Web server running together . The files in the DATA folder should be uploaded to the ESP32 internal repository via ESP32 Sketch Data Upload.

In other words, you have to upload twice separately from the sketch.

After uploading, A1S needs to set up Wi-Fi. At first run, 192.168.4.1 will operate in AP mode and insert the SSID PASSWORD of the router you want to use. Then, when you get an IP from the router and connect to the client IP, you will see the IC-706 Remote Control page, where you can enter the IP on the server side and the port to use. Audio:4000 / PTT:40010 / Serial:40020 and the client setup is complete.

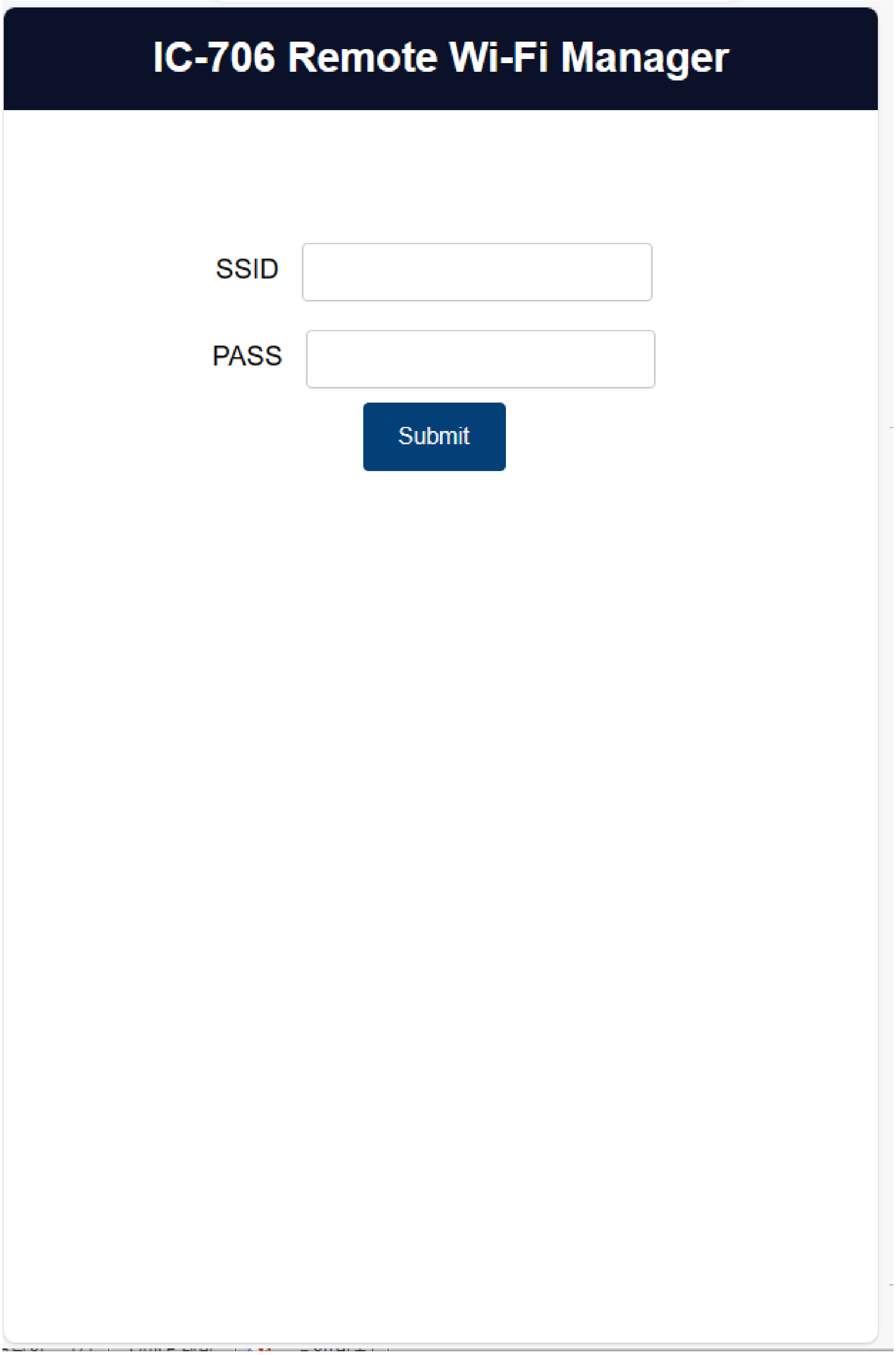


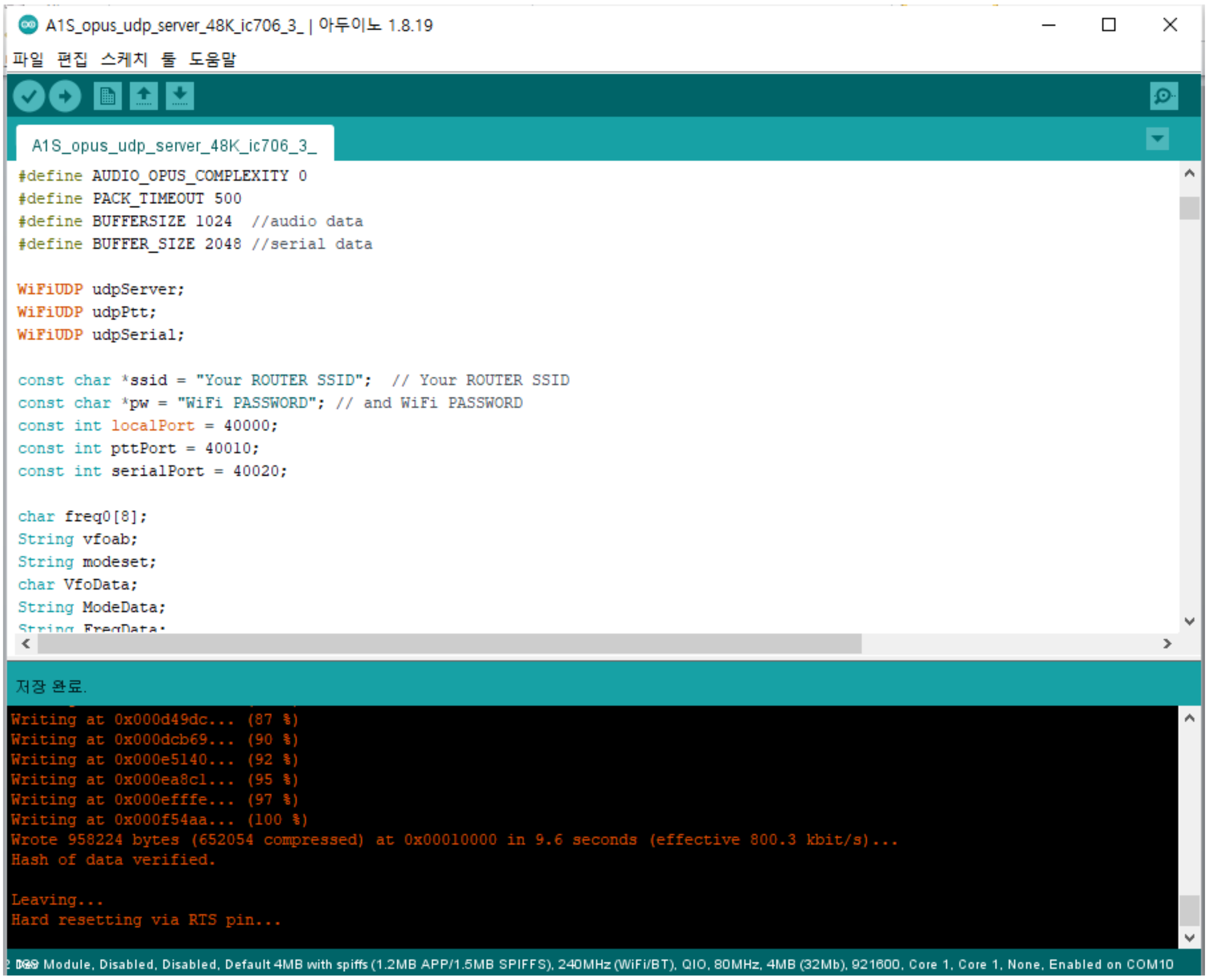
Figure 2 WiFi Management Page

Figure 3 Client Setup Page

Setting Arduino on the Server Side

On the server side, the wifi management server is not working. You must put ssid/password in the direct code. Make the server connect well to the router, find the server IP, put it in from the client side, and connect to each other . If you don't configure node servers and web controls, that's all.

To control the node server and web using Raspberry Pi, you can connect the uart microusb on the A1S side to the Raspberry Pi USB. (To get serial data)



Raspberry pie.

Just enter pi@raspberrypi:~ $ suo apt-get install nodejsnpm and you' re done with the server.

Copy and insert the shared wsAudioServer folder into the raspberry pie.

Running pi@raspberrypi:~/wsAudioServer $nodeserver.js completes server operation

Modify server.js file

audio_client.html server.js

```
const path = require("path");
const express = require("express");
const WebSocket = require("ws");
const { SerialPort } = require('serialport')
const { ReadlineParser } = require('@serialport/parser-readline')
const app = express();
const { spawn } = require('child_process');



const WS_PORT = process.env.WS_PORT || 40040;
const HTTP_PORT = process.env.HTTP_PORT || 8080;

const serialPortPath = '/dev/ttyUSB0';
const baudRate = 115200;
const serialPort = new SerialPort({ path: serialPortPath, baudRate: baudRate }, (err) => {
  if (err) {
    console.error(`Error opening serial port ${serialPortPath} at baud rate ${baudRate}:`, err);
  } else {
    console.log(`Serial port ${serialPortPath} opened at baud rate ${baudRate}`);
  }
});

const parser = serialPort.pipe(new ReadlineParser({ delimiter: '\r\n' }))
const wsServer = new WebSocket.Server({ port: WS_PORT }, () =>
  console.log(`WS server is listening at ws:localhost:${WS_PORT}`)
);

let connectedClients = [];
let arecordProcess = null;
let isPaused = false;

wsServer.on("connection", (ws, req) => {
   const clientId = Math.random().toString(36).substring(7);
  console.log(`Client connected: ${clientId}`);

  connectedClients.push({ id: clientId, socket: ws });

  ws.on("message", (message) => {
     connectedClients.forEach((client, i) => {
```

const WS_PORT = process.env.WS_PORT || || 40040 ;//audio socket port

const HTTP_PORT = process.env.HTTP_PORT || || 8080 ;//Webpage access port

ConstitutionalPortPath = '/dev/ttyUSB0 ';//A1S serial communication connection

The item is set to be correct by the user.

Audio_client.html 수정

```
        player = new PCMPlayer({
            inputCodec: 'Int16',
            channels: 1,
            //sampleRate: 16000,
            sampleRate: 44100,

        });

        canvas = document.getElementById('waveformCanvas');
        canvasContext = canvas.getContext('2d');

        const WS_URL = 'ws://61.77.62.48:40040';
        var ws = new WebSocket(WS_URL)
        ws.binaryType = 'arraybuffer'

        ws.addEventListener('message', function (event) {
           const data = event.data;
           if(playon){
           drawWaveform(data);
           player.feed(data);
           }

        });

        ws.onmessage = function (event) {
        data = event.data;


        if (data.startsWith("RIGDATA")) {
        var str = data;
        var freqdata = str.substr(7,10);
        var vfodata = str.substr(17,1);
        var modedata = str.substr(18,4);
        var menu = str.substr(22,2);
        var f1 = str.substr(24,3);
        var f2 = str.substr(27,3);
        var f3 = str.substr(30,3);
        var mch = str.substr(33,2);

```

const WS_URL = 'ws://61.77.62.48:40040'; The item sets its own Raspberry Pi IP .

In Raspberry Pi, the audio signal is taken from IC-706 and the microphone input is put in . (for transmitting audio from web pages)

Also, control signals must be imported from A1S (A1S is USB connected to Raspberry Pi, so it is a connection to exchange serial data).

Web control screen yet (signal display and frequency not changed improvements)

Node.js server needs to be rerun periodically due to intermittent down (improvements)

Hardware Connections