Chris Wren

N4VDR  n4vdr@wrenal.com

N4VDR fox manual

Table of Contents

[Concept 3](#_Toc34992593)

[Connections 3](#_Toc34992594)

[FCC Regulations 3](#_Toc34992595)

[Notes: 3](#_Toc34992596)

[Operation 4](#_Toc34992597)

[DTMF Commands 4](#_Toc34992598)

[1) Enabled Timed Transmit 4](#_Toc34992599)

[2) Speed up CW by 5 WPM 5](#_Toc34992600)

[3) Set Unit to Novice Mode 5](#_Toc34992601)

[4) Disabled Timed Transmit 5](#_Toc34992602)

[5) Slow down CW by 5 WPM 5](#_Toc34992603)

[6) Set Unit to Advanced Mode 5](#_Toc34992604)

[7) Transmit 7.5 Second Tone Burst 5](#_Toc34992605)

[Schematic 6](#_Toc34992606)

[Code 6](#_Toc34992607)

[Notes/Credits 25](#_Toc34992608)

# Concept

This unit is designed to function with a Baofeng UV-5R handheld radio, or other similar radio that uses the same handmic pinouts.

The unit is designed to transmit a CW signal on a random schedule.

# Connections

Plug the hand mic connector into the HT and the 4-ring 3.5mm jack into the fox. Make sure the 9VDC battery is connected inside the fox.

# FCC Regulations

This code is adjusted to comply with parts:

47 CFR Part 97 - AMATEUR RADIO SERVICE  
 Subpart A - General Provisions  
 § 97.3 Definitions  
 § 97.5 Station license required  
 § 97.7 Control operator required  
 Subpart B - Station Operation Standards  
 § 97.101 General standards  
 § 97.105 Control operator duties  
 § 97.109 Station control  
 § 97.111 Authorized transmissions  
 § 97.119 Station identification  
 Subpart C - Special Operations  
 § 97.201 Auxiliary station  
 § 97.203 Beacon station

## Notes:

1. Per 97.201, below frequencies are **NOT** allowed:
   1. 144.0-144.5 MHz
   2. 145.8-146.0 MHz
   3. 219-220 MHz
   4. 222.00-222.15 MHz
   5. 431-433 MHz
   6. 435-438 MHz
2. Per 97.203, below frequencies are allowed
   1. 28.20-28.30 MHz
   2. 50.06-50.08 MHz
   3. 144.275-144.300 MHz
   4. 222.05-222.06 MHz
   5. 432.300-432.400 MHz
   6. 33 cm and shorter wavelength bands

# Operation

1. Ensure the proper connections are made
2. Ensure the fox has power
3. Turn on the radio and set volume to approximately 50%
4. Set fox in “hiding” position and walk away.

# DTMF Commands

|  |  |
| --- | --- |
| DTMF Value sent | Action |
| 1 | Enabled Timed Transmit |
| 2 | Speed up CW by 5 WPM |
| 3 | Set Unit to Novice Mode |
| 4 | Disable Timed Transmit |
| 5 | Slow down CW by 5 WPM |
| 6 | Set Unit to Advanced Mode |
| 7 | Transmit 7.5 second Tone Burst |

## Enabled Timed Transmit

This command sets the “automatic” transmission. Depending on the mode of the unit (novice vs advanced), the unit will send the beacon string of “CQ N4VDR/FOX” at random intervals.

## Speed up CW by 5 WPM

At bootup, the CW speed is set to 20 WPM. This can be adjusted up or down depending on the speed desired. Minimum speed is 5 WPM, maximum speed is 30 WPM.

## Set Unit to Novice Mode

Novice mode sets the fox to transmit the beacon string a total of 4 times. The timing between transmissions is random with the minimum time being 1 minute and 4 minutes.

## Disabled Timed Transmit

If for some reason the fox needs to be silenced completely, this will disable all transmissions from the fox until either the fox is rebooted or the “enable timed transmit” command is sent to the fox.

## Slow down CW by 5 WPM

At bootup, the CW speed is set to 20 WPM. This can be adjusted up or down depending on the speed desired. Minimum speed is 5 WPM, maximum speed is 30 WPM.

## Set Unit to Advanced Mode

Advanced mode sets the fox to transmit the beacon string a total of 2 times. The timing between transmissions is random with the minimum time being 5 minutes and 10 minutes

## Transmit 7.5 Second Tone Burst

This will cause the fox to transmit for 7.5 seconds a 700 hz tone burst.

All DTMF commands will be acknowledged by the fox. When a DTMF command is received, the fox will transmit:  
 “[DTMF number] QSL”

Example, if the fox receives a DTMF command 7, the fox will transmit:

“7 qsl” followed by a 7.5 second tone.

# Schematic

A map with text

Description automatically generated

# Code

#include <DTMF.h>

int advLed = 2; // Dig I/O

int PTT = 12; // Dig I/O MOSI

int audio = 13; // Dig I/O SCK

int sensorPin = A0; // Analog In 0

char beaconString[] = "CQ N4VDR/FOX";

char cmdValue1[] = "1 qsl";

char cmdValue2[] = "2 qsl";

char cmdValue3[] = "3 qsl";

char cmdValue4[] = "4 qsl";

char cmdValue5[] = "5 qsl";

char cmdValue6[] = "6 qsl";

char cmdValue7[] = "7 qsl";

const int freq = 700;

const float n = 128.0;

const float sampling\_rate = 8926.0

float d\_mags[8];

const int ditLenArray[6] = {240,120,80,60,48,40};

DTMF dtmf = DTMF(n, sampling\_rate);

bool txEn = false;

bool rxEn = true;

bool novice = true;

bool storage = true;

int pause = 60000;

int rMin = 60000;

int rMax = 240000;

int wpmCW = 3;

long prevMilli = millis();

int cmd;

int prevCmd;

//============ Setup Function ============

void setup() {

pinMode(PTT, OUTPUT);

pinMode(advLed, OUTPUT);

digitalWrite(PTT, HIGH);

randomSeed(analogRead(A0));

txEn = true;

digitalWrite(advLed, !novice);

}

//============ Loop Function ============

void loop(){

if (rxEn == true){

dtmfListen();

}

if ((millis() - prevMilli >= pause)&&(txEn == true)){

if (novice == true){

beacon();

space();

beacon();

} else {

beacon();

}

pause = random(rMin,rMax);

#ifdef DEBUG\_ON

Serial.print(F("Delay time set to: "));

Serial.println(pause/1000);

#endif

prevMilli = millis();

}

}

//============ DTMF Listen Function ============

void dtmfListen() {

dtmf.sample(sensorPin);

dtmf.detect(d\_mags, 506);

cmd = dtmf.button(d\_mags, 1800.) - 48;

if (cmd > 0){

prevCmd = cmd;

switch (cmd){

case 1:

dtmfOne();

break;

case 2:

dtmfTwo();

break;

case 3:

dtmfThree();

break;

case 4:

dtmfFour();

break;

case 5:

dtmfFive();

break;

case 6:

dtmfSix();

break;

case 7:

dtmfSeven();

break;

/\*

case 8:

dtmfEight();

break;

case 9:

dtmfNine();

break;

\*/

default:

dtmfError();

break;

}

}

delay(1);

}

//============ Beacon Function ============

void beacon() {

rxEn = false;

digitalWrite(PTT, LOW);

delay(1500);

for (int c = 0; c < 2; c++) {

for (int i = 0; i < sizeof(beaconString) - 1; i++) {

char tmpChar = beaconString[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(1500);

}

delay(500);

digitalWrite(PTT, HIGH);

rxEn = true;

}

//============ DTMF 1 Function ============

void dtmfOne() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < sizeof(cmdValue1) - 1; i++) {

char tmpChar = cmdValue1[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(500);

digitalWrite(PTT, HIGH);

txEn = true;

rxEn = true;

cmd = 0;

}

//============ DTMF 2 Function ============

void dtmfTwo() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < sizeof(cmdValue2) - 1; i++) {

char tmpChar = cmdValue2[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(500);

digitalWrite(PTT, HIGH);

if (wpmCW < 5) {

wpmCW++;

} else {

wpmCW = 5;

}

rxEn = true;

txEn = storage;

cmd = 0;

}

//============ DTMF 3 Function ============

void dtmfThree() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < sizeof(cmdValue3) - 1; i++) {

char tmpChar = cmdValue3[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(500);

digitalWrite(PTT, HIGH);

novice = true;

digitalWrite(advLed, LOW);

rMin = 60000;

rMax = 240000;

pause = random(rMin,rMax);

rxEn = true;

txEn = storage;

cmd = 0;

}

//============ DTMF 4 Function ============

void dtmfFour() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < sizeof(cmdValue4) - 1; i++) {

char tmpChar = cmdValue4[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(500);

digitalWrite(PTT, HIGH);

txEn = false;

rxEn = true;

cmd = 0;

}

//============ DTMF 5 Function ============

void dtmfFive() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < sizeof(cmdValue5) - 1; i++){

char tmpChar = cmdValue5[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(500);

digitalWrite(PTT, HIGH);

if (wpmCW > 0) {

wpmCW--;

} else {

wpmCW = 0;

}

rxEn = true;

txEn = storage;

cmd = 0;

}

//============ DTMF 6 Function ============

void dtmfSix() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < sizeof(cmdValue6) - 1; i++) {

char tmpChar = cmdValue6[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(500);

digitalWrite(PTT, HIGH);

novice = false;

digitalWrite(advLed, HIGH);

rMin = 300000;

rMax = 600000;

pause = random(rMin,rMax);

rxEn = true;

txEn = storage;

cmd = 0;

}

//============ DTMF 7 Function ============

void dtmfSeven() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < sizeof(cmdValue7) - 1; i++){

char tmpChar = cmdValue7[i];

tmpChar = toLowerCase(tmpChar);

getChar(tmpChar);

}

delay(500);

tone(audio, freq);

delay(7500);

noTone(audio);

delay(500);

digitalWrite(PTT, HIGH);

txEn = storage;

rxEn = true;

cmd = 0;

}

//============ DTMF ERROR Function ============

void dtmfError() {

storage = txEn;

txEn = false;

rxEn = false;

delay(1500);

digitalWrite(PTT, LOW);

delay(1500);

for (int i = 0; i < 3; i++){

tone(audio, 350);

delay(500);

noTone(audio);

delay(500);

}

delay(500);

digitalWrite(PTT, HIGH);

txEn = storage;

rxEn = true;

cmd = 0;

}

//%%%%% EVERYTHING BELOW HERE IS TO CREATE THE MORSE CODE %%%%%

//============ CW Dit Function ============

void dit(){

tone(audio, freq);

delay(ditLenArray[wpmCW]);

noTone(audio);

delay(ditLenArray[wpmCW]);

}

//============ CW Dah Function ============

void dah(){

tone(audio, freq);

delay(ditLenArray[wpmCW]\*3);

noTone(audio);

delay(ditLenArray[wpmCW]);

}

//============ CW End of Character Function ============

void endChar()

{

delay(ditLenArray[wpmCW]\*3);

}

//============ CW End of Character Function ============

void space()

{

delay(ditLenArray[wpmCW]\*6);

}

//============ CW End of Word Function ============

void endWord()

{

delay(ditLenArray[wpmCW]\*7);

}

//============ Convert Char to CW Function ============

void getChar(char tmpChar)

{

// Take the passed character and use a switch case to find the morse code for that character

// A thorugh Z

switch (tmpChar) {

case 'a':

dit();

dah();

endChar();

break;

case 'b':

dah();

dit();

dit();

dit();

endChar();

break;

case 'c':

dah();

dit();

dah();

dit();

endChar();

break;

case 'd':

dah();

dah();

dit();

endChar();

break;

case 'e':

dit();

endChar();

break;

case 'f':

dit();

dit();

dah();

dit();

endChar();

break;

case 'g':

dah();

dah();

dit();

endChar();

break;

case 'h':

dit();

dit();

dit();

dit();

endChar();

break;

case 'i':

dit();

dit();

endChar();

break;

case 'j':

dit();

dah();

dah();

dah();

endChar();

break;

case 'k':

dah();

dit();

dah();

endChar();

break;

case 'l':

dit();

dah();

dit();

dit();

endChar();

break;

case 'm':

dah();

dah();

endChar();

break;

case 'n':

dah();

dit();

endChar();

break;

case 'o':

dah();

dah();

dah();

endChar();

break;

case 'p':

dit();

dah();

dah();

dit();

endChar();

break;

case 'q':

dah();

dah();

dit();

dah();

endChar();

break;

case 'r':

dit();

dah();

dit();

endChar();

break;

case 's':

dit();

dit();

dit();

endChar();

break;

case 't':

dah();

endChar();

break;

case 'u':

dit();

dit();

dah();

endChar();

break;

case 'v':

dit();

dit();

dit();

dah();

endChar();

break;

case 'w':

dit();

dah();

dah();

endChar();

break;

case 'x':

dah();

dit();

dit();

dah();

endChar();

break;

case 'y':

dah();

dit();

dah();

dah();

endChar();

break;

case 'z':

dah();

dah();

dit();

dit();

endChar();

break;

// 0 (Zero) Through 9

case '0':

dah();

dah();

dah();

dah();

dah();

endChar();

break;

case '1':

dit();

dah();

dah();

dah();

dah();

endChar();

break;

case '2':

dit();

dit();

dah();

dah();

dah();

endChar();

break;

case '3':

dit();

dit();

dit();

dah();

dah();

endChar();

break;

case '4':

dit();

dit();

dit();

dit();

dah();

endChar();

break;

case '5':

dit();

dit();

dit();

dit();

dit();

endChar();

break;

case '6':

dah();

dit();

dit();

dit();

dit();

endChar();

break;

case '7':

dah();

dah();

dit();

dit();

dit();

endChar();

break;

case '8':

dah();

dah();

dah();

dit();

dit();

endChar();

break;

case '9':

dah();

dah();

dah();

dah();

dit();

endChar();

break;

case '/': // Slash

dah();

dit();

dit();

dah();

dit();

endChar();

break;

case '.': // Period

dit();

dah();

dit();

dah();

dit();

dah();

endWord();

break;

case '-': // Dash

dah();

dit();

dit();

dit();

dit();

dah();

endChar();

break;

case ',': // Comma

dah();

dah();

dit();

dit();

dah();

dah();

endChar();

break;

case '?': // Question Mark

dit();

dit();

dah();

dah();

dit();

dit();

endChar();

break;

case '!': // Exclamation Mark

dah();

dit();

dah();

dit();

dah();

dah();

endChar();

break;

case '"': // Quotes

dit();

dah();

dit();

dit();

dah();

dit();

endChar();

break;

case ':': // Colon

dah();

dah();

dah();

dit();

dit();

dit();

endChar();

break;

case '@': // At

dit();

dah();

dah();

dit();

dah();

dit();

endChar();

break;

default:

// If a matching character was not found it will default to a blank space

space();

}

}

# Notes/Credits

The idea and concept for this unit was inspired by Gary Griffin - WT4Y and the North East Georgia Amateur Radio Club – NE4GA

Gary’s original design and concept can be seen at <http://www.ne4ga.org/projects/arduinocontrolledfox>

This project is an open source project. The schematics, code, libraries, and all associated files can be found on github at <https://github.com/vader7071/Arduino_Fox>