# Introduction

For Christmas I got a Baofeng UV5R radio, which is a real cheep ($40) ham radio. Since I did not have a ham license, I started by simply programming it to be a police scanner. However, I soon got bored of just listening around and wanted to see what can be done with this radio. Since its a 4W radio, I thought it would be cool to try and send serial data over a few miles. After searching around a bit, I found that it can be programmed with an open source program called CHIRP. I was originally hoping that I could change the firmware, but after looking at the CHIRP code I realized that you can only program the channels in the radio, and a few settings. I even attempted to write a simple python script (derived from the chirp program), to see if other commands are available, but with no success. Since the schematics were available, I thought I could just try to interface with the cpu directly.

{code}
#Python program for Simple uv5r interface derived from CHIRP
import struct
import time
import serial
from chirp import chirp_common, errors, util, directory, memmap

UV5R_MODEL_291 = "x01xBBxFFx20x12x07x25"

def _read_block(ser, start, size):
msg = struct.pack(">BHB", ord("S"), start, size)
ser.write(msg)

answer = ser.read(4)
if len(answer) != 4:
raise errors.RadioError("Radio refused to send block 0x%04x" % start)

cmd, addr, length = struct.unpack(">BHB", answer)
if cmd != ord("X") or addr != start or length != size:
print "Invalid answer for block 0x%04x:" % start
print "CMD: %s ADDR: %04x SIZE: %02x" % (cmd, addr, length)
raise errors.RadioError("Unknown response from radio")

chunk = ser.read(0x40)
if not chunk:
raise errors.RadioError("Radio did not send block 0x%04x" % start)
elif len(chunk) != size:
print "Chunk length was 0x%04i" % len(chunk)
raise errors.RadioError("Radio sent incomplete block 0x%04x" % start)

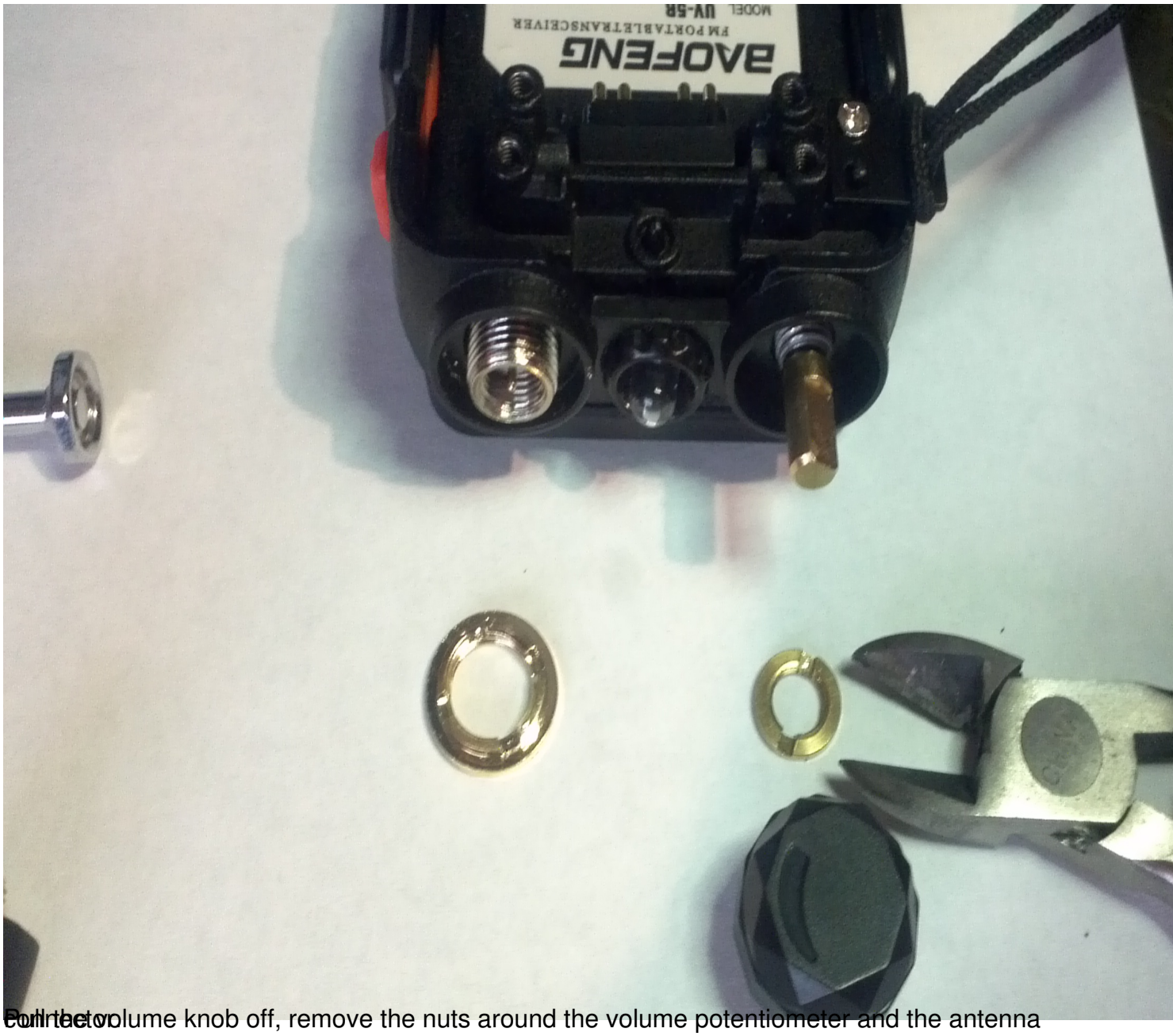
ser.write("x06")

```
 ack = ser.read(1)
 if ack != "x06":
 raise errors.RadioError("Radio refused to send block 0x%04x" % start)

 return chunk

 ser = serial.Serial(port="/dev/ttyUSB0", baudrate=9600, timeout=0.5)
 ser.setTimeout(1)
 ser.write(UV5R_MODEL_291)
 ack = ser.read(1)
 ser.write("x02")
 ident = ser.read(8)
 print "Ident:n%s" % util.hexprint(ident)

 ser.write("x06")
 ack = ser.read(1)
 print "Ack %s" % util.hexprint(ack);
 data = ""
 for i in range(0, 0x8192, 0x40):
 data += _read_block(ser, i, 0x40)

 print "Data:n%s" % util.hexprint(data)
{/code}
```

# Opening the Radio

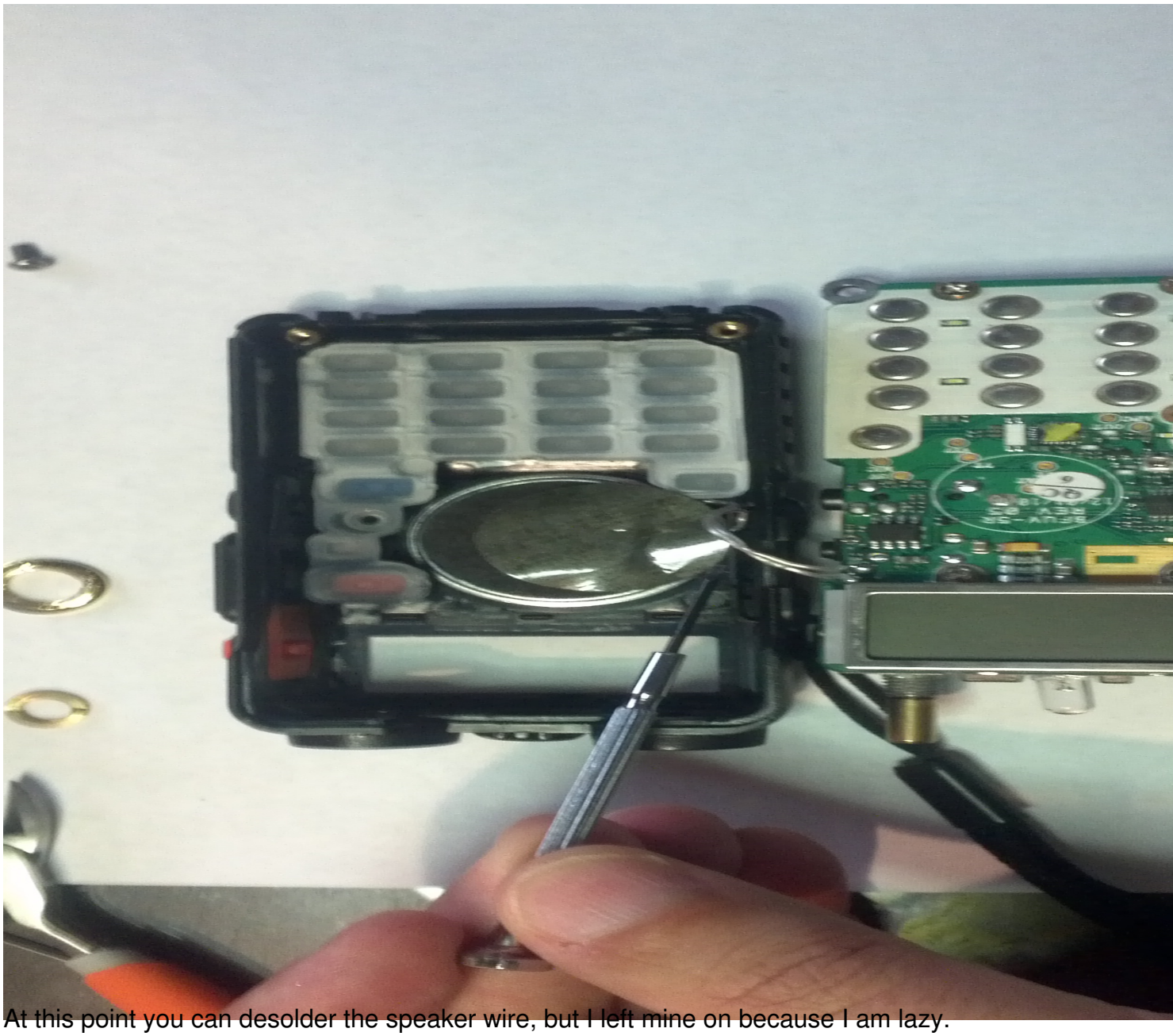
This was not too difficult. I found some [information online] from someone who needed to change the TX modulations.


You see the battery and the spring. Remove the two screws on the back of the radio.

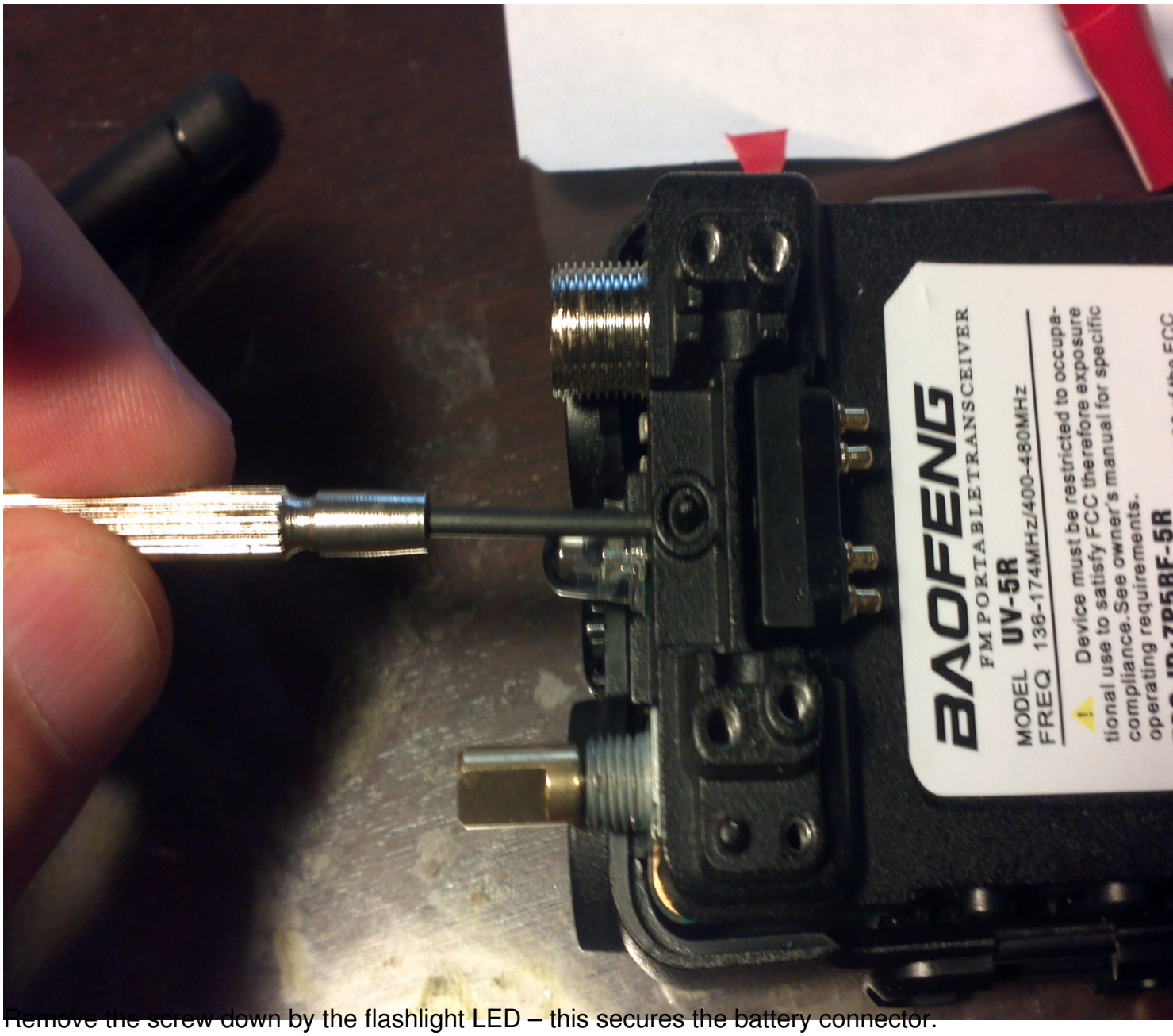



Pull the volume knob off, remove the nuts around the volume potentiometer and the antenna connector.


Pry the speaker wire for the radio a bit, and pull that section back and away from the case (watch out for the black wire).

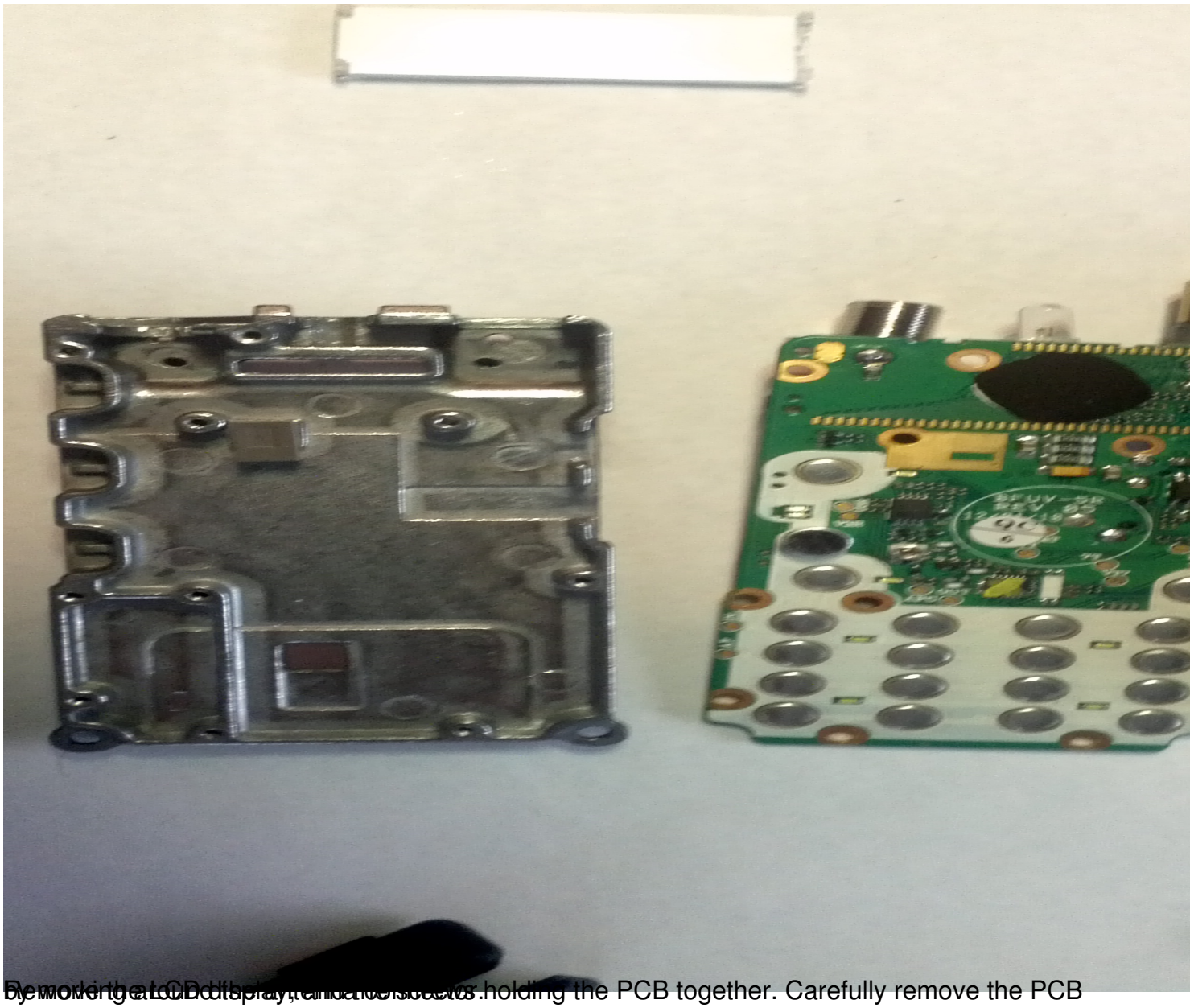


At this point you can desolder the speaker wire, but I left mine on because I am lazy.

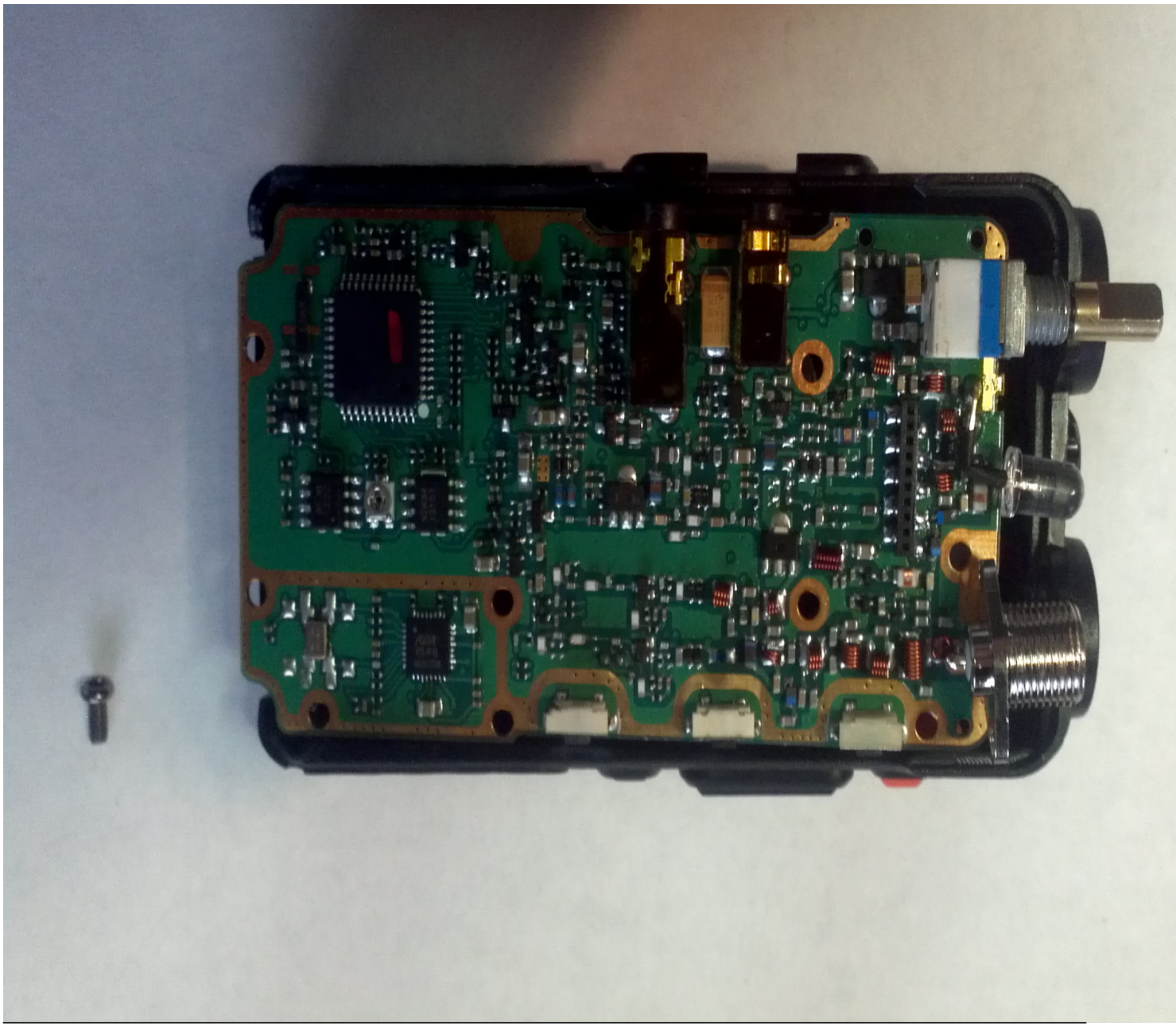



Remove the screw down by the flashlight LED – this secures the battery connector.


Remove the battery connector



By removing the LCD display, and the screws holding the PCB together. Carefully remove the PCB



# Hacking the radio

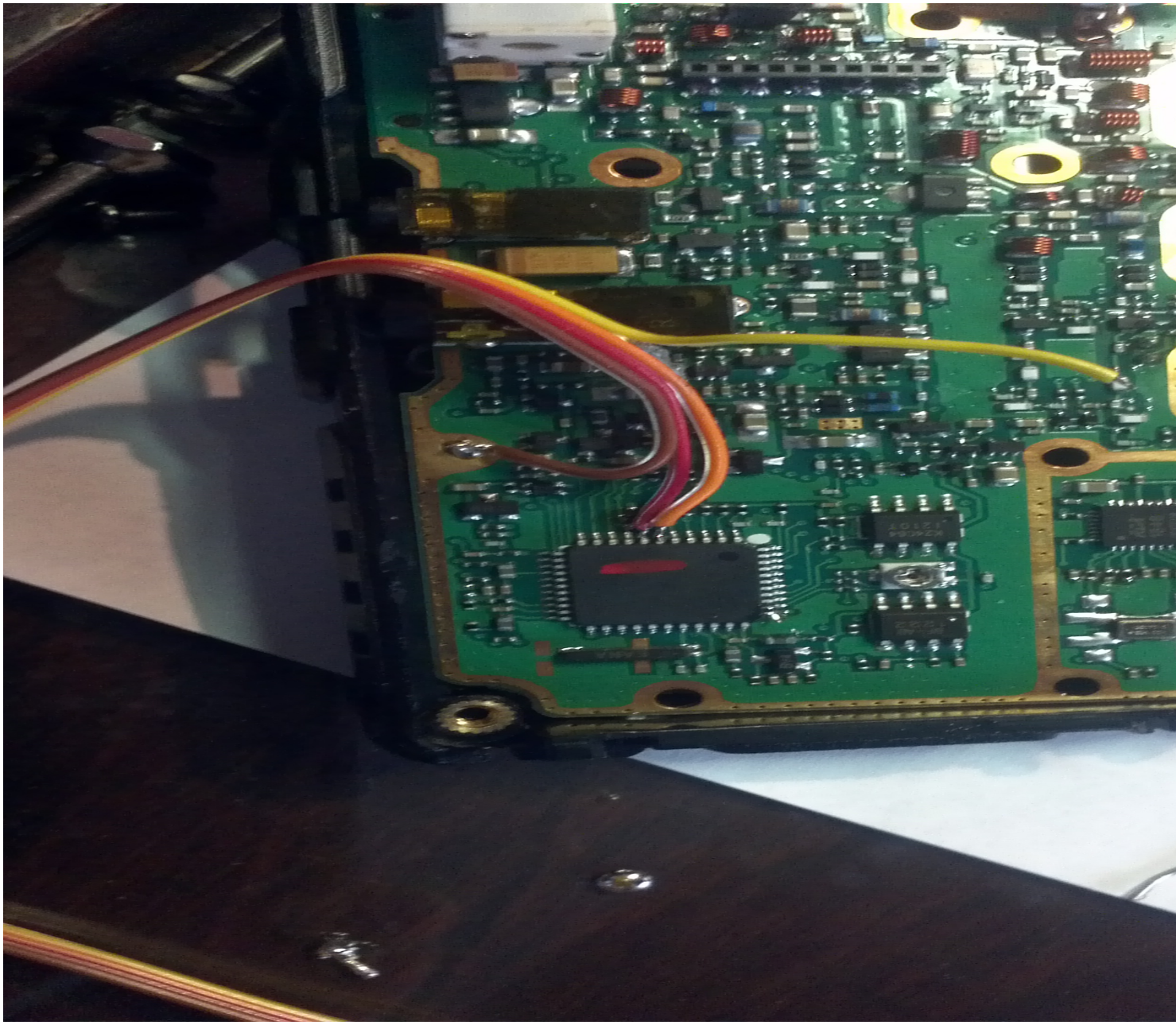
Before trying to change anything I posted on the baofeng_uv5r yahoo groups to see what I would need to insure that I will not radiate any RF energy. I found out that I could not just remove the antenna, and would need to place a dummy load. Per James Hall suggestion I ordered a dummy load from here
qrpkits
.

As soon as I got the radio, I signed up to take the license test. Unfortunately, the next one is on Jan 20. I can not wait to get licensed, so I can finally transmit and test this on the HAM frequencies. I also have not done anything with TX on this radio yet, which I am so attempted to

try to see if that works.So for now I am just going to try to RX until I get my license.

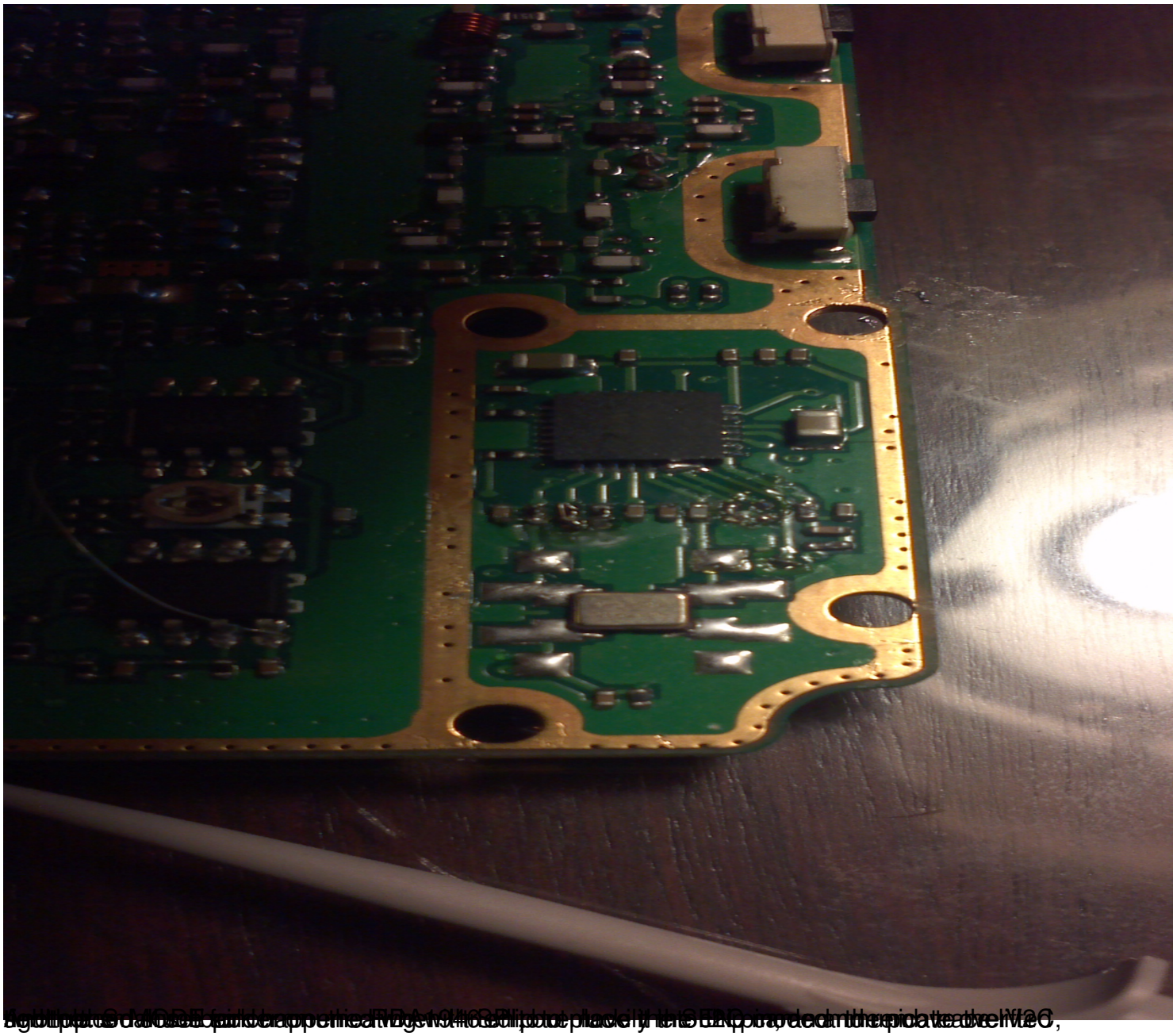
Looking at the schematics, there are two main ICs on the radio. The first one is RDA1846 which does all the radio communications, and the other is an EM78P568 microcontroller which controls the RDA1846 , LCD, keypad, etc.

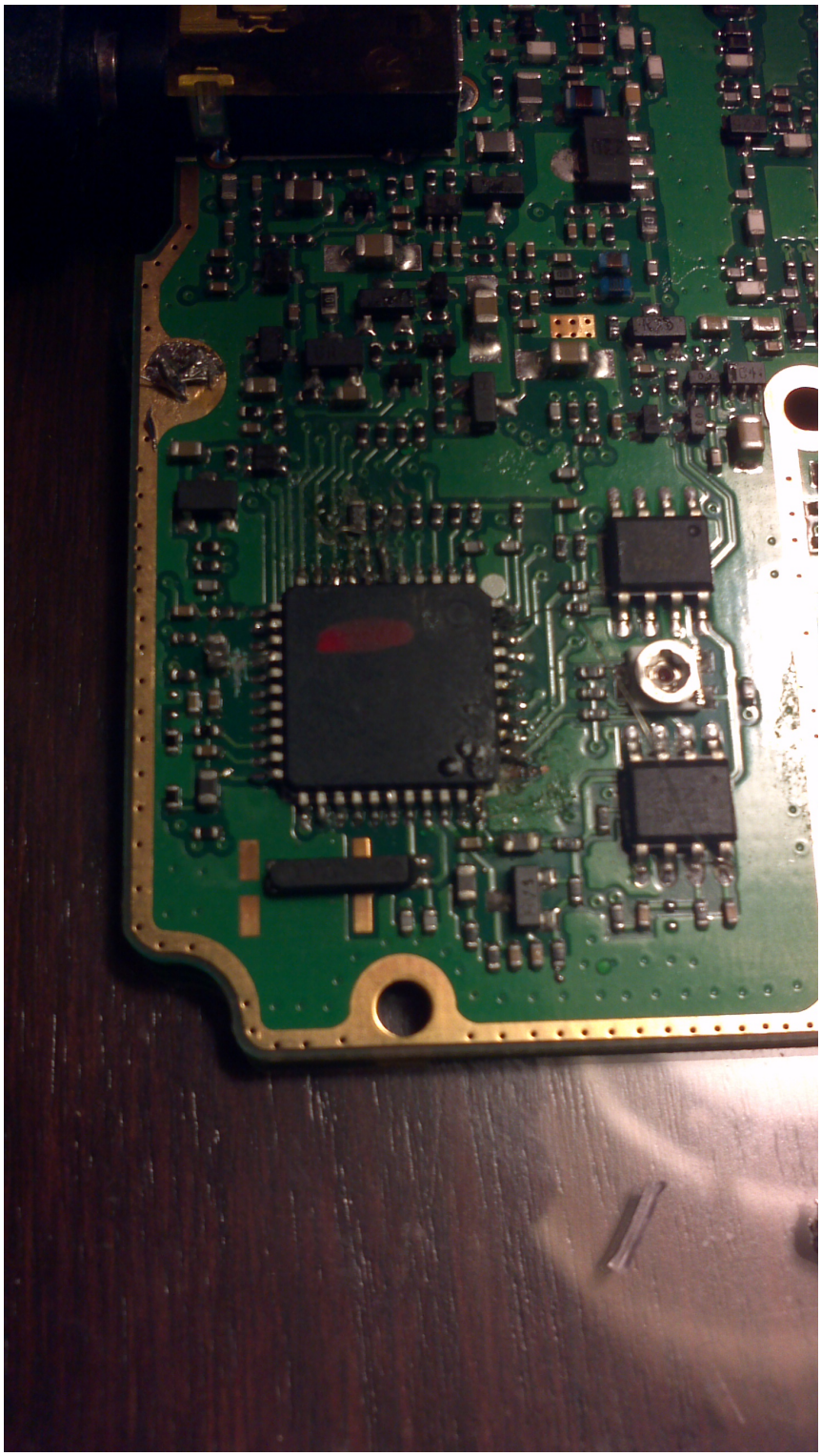
I first tried to solder some wires directly onto the RDA1846 TX/RX pins and monitor them to see what the radio sends the chip.

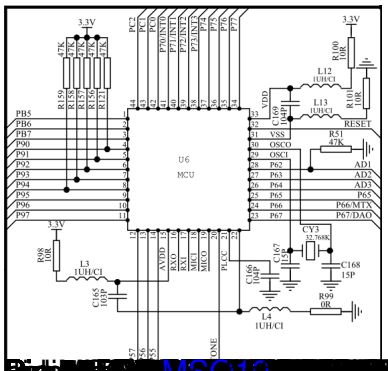


First, I was hoping that the microcontroller will not communicate with the RDA1846 under various conditions (like switching to the  RDA5802 FM radio) so I can also send commands to it. However, I monitored the SCLK and SDIO and it looks like its under constant communication. Since P93 and P92 are shared with the LCD (DB6, and DB7) as well as the keypad, the uv5r MCU is constantly setting/reading these pins. So their code probably has a loop setting the LCD, reading the keypad, and then transmitting serial commands to the RDA1846.

To resolved this I lifted P93 and P92 legs from the MCU (SCLK and SDIO) and solder two wires to it. I also soldered another line to P77 which is the SEN line. From the schematics, it looks like the MODE pin is tied to V+, which means it's communicating via SPI instead of I2C and the SEN line is used for ~EN.
I hooked up an arduino to the pins and tried to communicate over SPI, with no success. However, I should have not rushed into things, and noticed that the chip is **3.3V and not 5V** (I was looking in their programming manual the whole time, and should have read the datasheet first).



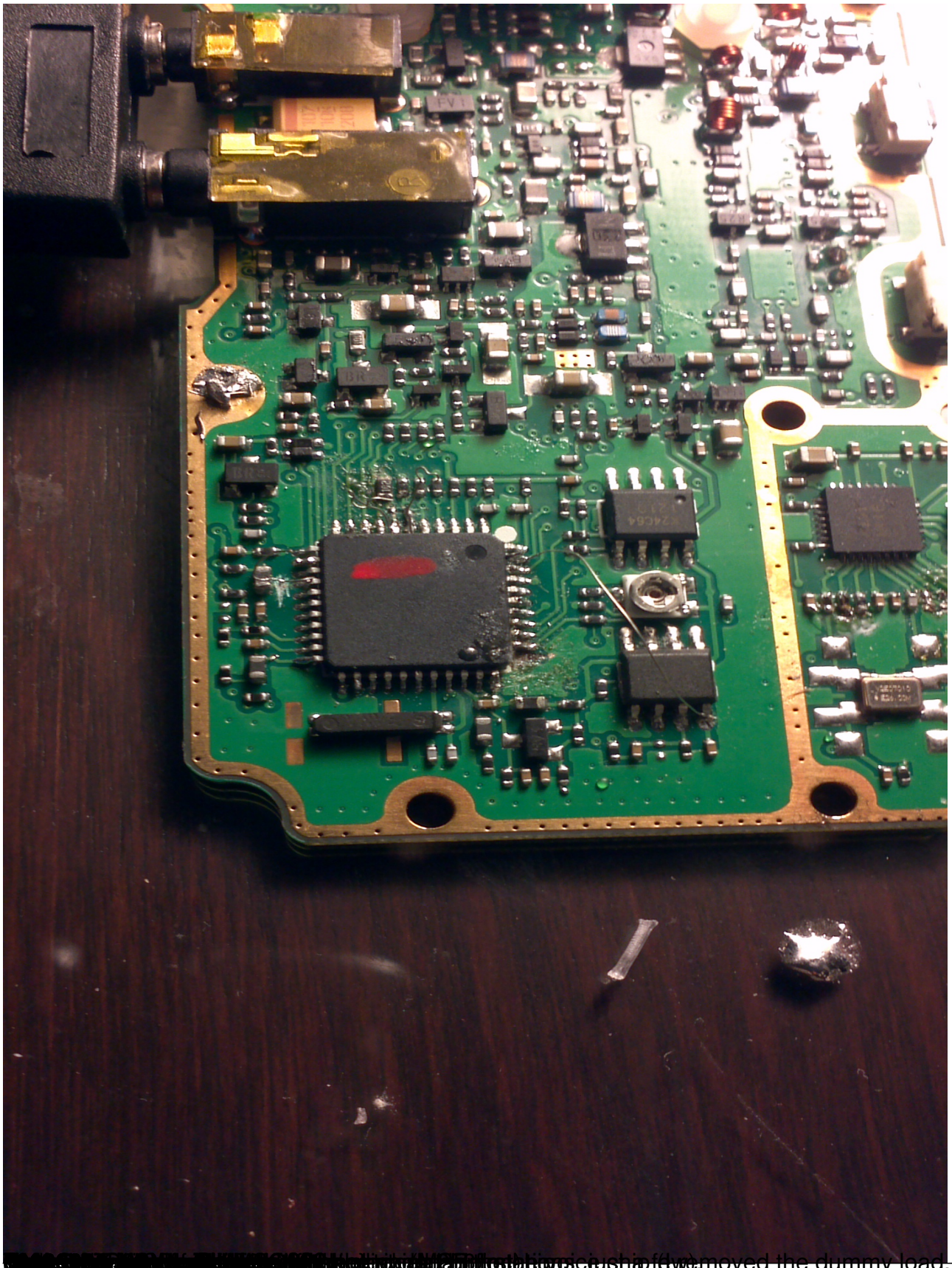





MSC12

# Interfacing with the voice chip

I figured out how to interface with the voice chip. Its very easy. In the process, I also found out a good way of experimenting with the MCU in place. I cut the Vdd traces going into the MCU and added jumper wires instead (there are two vdd lines going into the MCU with the trace exposed). This allows me to turn off the cpu whenever I want to send my own signals, and turn on the cpu whenever I want to see what the cpu is doing. I also soldered very thin wires (I used a single strand from a multi strand wire) on the the legs of the MCU so I can attach the probes and sniff the logic data or replay the data.



# Interfacing with the RDA5802

I was also able to interface with the  RDA5802 chip (the FM radio receiver).

The interface is I2C, and I was able to hook up an Arduino directly. Even though the chip is 3.3, you can hook it up with two 4.7K resistors tied to the 3.3 per I2CBi-directionalLevelShifter

However, the RDA5802 datasheet does not seem to have the registers that the radio is setting. I was able to capture the basic data that the uv5r is sending and play them back.

Here is the protocol (number are in hexadicimal).
When the radio boots up it sends: i2Start 0x10 w 0xD2 0x00 STOP  i2cStart 0x10 w 0xD2 0x00 0x3D 0xD8 STOP
When the radio switches to radio FM mode is sends: i2Start 0x10 w 0x90 0x3 0x00 0x18 STOP i2cStart 0x10 w 0xD0 0x01 0x2A 0x18 STOP
Changing the channel the radio sends: i2cStart 0x10 w 0xD0 0x01 0x27 0x98 stop //for 91.8FM
So the channels seem to be
91.7FM is 0x2758
91.8FM is 0x2798
91.9FM is 0x27D8
However, when I tuned to some channel (I dont remember which freq it was) 0x28D4 a station comes in clear, when the freq is set to 0x28D5 all you hear is static, and the same for 0x28D6 and 0x28D7. However, when the freq is set to 0x28D8 the same station comes in clear. Stepping through the numbers the same channel will come in clear but the distance between the numbers grows larger and other channels are coming in between these numbers. So it looks like the stations are interleaved with one another. Any ideas why this is so? I think it might be harmonics.
I placed a youtube video showing the results.

{youtube}2LdqJCkUGS4{/youtube}

Here is the arduino code I was using to step though the channels
{code}
```
#include <Wire.h>
//  Arduino analog input 5 - I2C SCL
//  Arduino analog input 4 - I2C SDA
#define ADDRESS B0010000
void setup() {
 // ...
 Wire.begin();
 Serial.begin(9600);
 Wire.beginTransmission(ADDRESS);
 Wire.write(0x90);
 Wire.write(0x03);
 Wire.write(0x00);
 Wire.write(0x18);
 Wire.endTransmission();
}
int ch = 0x2718; //Start at a good channel
void loop()
```

```
{
 if (Serial.available() > 0) {
 int in = Serial.read();
 int mute = 0;
 switch (in)
 {
 case 113:   ch++;   break;  //q
 case 97:    ch--;   break;  //a
 case 'm':    mute=1;   break;
 }
 Serial.print("Channel: ");
 unsigned char chu = (ch >> 8) & 0xFF;
 unsigned char chl = ch&0xFF;
 Serial.println(ch, HEX);
 if (!mute)
 {
 Wire.beginTransmission(ADDRESS);
 Wire.write(0xD0);
 Wire.write(0x01);
 Wire.write(chu);
 Wire.write(chl);
 Wire.endTransmission();
 } else {
 Wire.beginTransmission(ADDRESS);
 Wire.write(0xD2);
 Wire.write(0x00);
 Wire.write(0x3D);
 Wire.write(0xD8);
 Wire.endTransmission();
 }
 }
}
{/code}
```

# Interfacing with the RDA1846

I was able to captured some data for how the radio is setup when its first powers up. Here is the sequence that the radio is sending the RDA1846 at power on.

The first bit is for read/!write then address and lastly data

0 0110000(0x30)  0000000000000001  soft_reset
0 0110000(0x30)  0000000000000100  pdn_reg same as pdn

0 0000100(0x04) 0000111111010000 clk_mode 24~28MHZ
0 0001011(0x0B) 0001101000010000 Not in Manual
0 0101011(0x2B) 0011001011001000 xtal_freq (13000/1000)*2=26MHz
0 0101100(0x2C) 0001100101100100 Adc clk freq: (6500/1000)*4=26MHz
0 0110001(0x31) 0011111111000000 Not in Manual
0 0110010(0x32) 0110001001111110 Not in Manual
0 0110011(0x33) 0000101011110010 Not in Manual
0 1000111(0x47) 0011101111101100 Not in Manual
0 1001111(0x4F) 0001000001000000 Not in Manual
0 1001110(0x4E) 0010100100111010 Not in Manual
0 1010110(0x56) 0000011001010010 Not in Manual
0 1101110(0x6E) 0000011000101101 Not in Manual
0 1110000(0x70) 0001100000011011 Not in Manual
0 1110001(0x71) 0110110000011110 Not in Manual
0 1111111(0xFF) 0000000000000001 Not in Manual
0 0000101(0x05) 0000000000011111 Not in Manual
0 1111111(0xFF) 0000000000000000 Not in Manual
0 0111100(0x3C) 0000101001111000 Tx voice signal from MIC
0 0111101(0x3D) 0010000000001011 Not in Manual
0 0011111(0x1F) 0001000000000001 gpio6 sq out, gpio0 css out/in/cmp
0 0001010(0x0A) 0000001101000000 1.01V pabias voltage
0 0000010(0x02) 0000011010011000 Not in Manual
0 1010100(0x54) 0001110101000000 gpio6 is sq only

0 0101001 (0x29) 0000000000111000 Freq high value
0 0101010 (0x2A) 0111101111000100 Freq low value
11100001111011111000100 = 3701700 Dec
3701700/(8*1000) = 462.7125MHz
The radio was tuned to FRS ch 7
0 0001111 (0x0F) 0011110100100100 Band Select 400-520MHz

Finally I managed to interface with the RDA1846 over i2c (I changed the protocol from spi to i2c, since its simpler to interface with the arduino, and for the fact that I ripped up the PDN pad). For a while I was able to interface with the RDA1846, but I was only receiving static when the sq was open. It took me a while, but then I remembered that they are switching the RX signal on an off to save on battery, and since I have uv5r mcu disabled, it did not turn on the RX circuit. I tied pin PC1 (RX POW) to 3.3v and everything seemed to work. I tested the radio by tuning it to a FRS channel 7 and using one of my FRS radio, I sent DTMF codes. I know this is not completely legit, but I think I am not technically doing anything wrong (the FRS radio I bought at radio shack does not need a license, and I am only using the UV5r as a receiver. Please let me know if I am correct about this).

I also wrote an arduino code to configure registers via the serial terminal. It looks like the UV5R mcu monitors the sq pin and when it is on, it sends a RX on to the RDA1846. So for now I was

basically just turning on the RX and turning it off. Here is a video of the radio receiving the DTMF tones on the FRS radio service.

{youtube}S71XOMIpkog{/youtube}

Here is the configuration sequence I was using for the chip (the format is register value, register value, ....)
0x30 0x0004, 0x04 0x0FD0, 0x0B 0x1A10, 0x2B 0x32C8, 0x2C 0x1964, 0x31 0x3FC0, 0x32 0x627E, 0x33 0x0AF2, 0x47 0x3BEC, 0x4F 0x1040, 0x4E 0x293A, 0x56 0x0652, 0x6E 0x062D, 0x70 0x181B, 0x71 0x6C1E, 0xFF 0x0001, 0x05 0x001F, 0xFF 0x0000, 0x3C 0x0A78, 0x3D 0x200B, 0x1F 0x1001, 0x0A 0x0340, 0x02 0x0698, 0x54 0x1D40, 0x29 0x0038, 0x2A 0x7BC4, 0x0F 0x3D24

To turn on the TX
0x30 0x3826
To turn off the TX
0x30 0x3806

Here is the arduino code
{codecitation class="brush:cpp" width="" }

```
#include <Wire.h>
#define ADDRESS B1110001
void setup() {
 Wire.begin();
 Serial.begin(9600);
}
byte getVal(char c)
{
 if(c >= '0' && c <= '9')
 return (byte)(c - '0');
 else
 return (byte)(c-'A'+10);
}
void loop()
{
 if (Serial.available() > 0)
 {
 unsigned char d = Serial.read();
 if (d == 'S')
 {
 int i=0;
 char data[8];
 while(i < 8)
 if (Serial.available() > 0)
 data[i++] = Serial.read();
```

```
 unsigned char address = getVal(data[1]) + (getVal(data[0]) << 4);
 unsigned char dataU = getVal(data[4]) + (getVal(data[3]) << 4);
 unsigned char dataL = getVal(data[7]) + (getVal(data[6]) << 4);
 Serial.println(address, HEX);
 Serial.println(dataU, HEX);
 Serial.println(dataL, HEX);
 Serial.println();
 Wire.beginTransmission(ADDRESS);
 Wire.write(address);
 Wire.write(dataU);
 Wire.write(dataL);
 Wire.endTransmission(1);
 }
 }
}
```

{/codecitation}

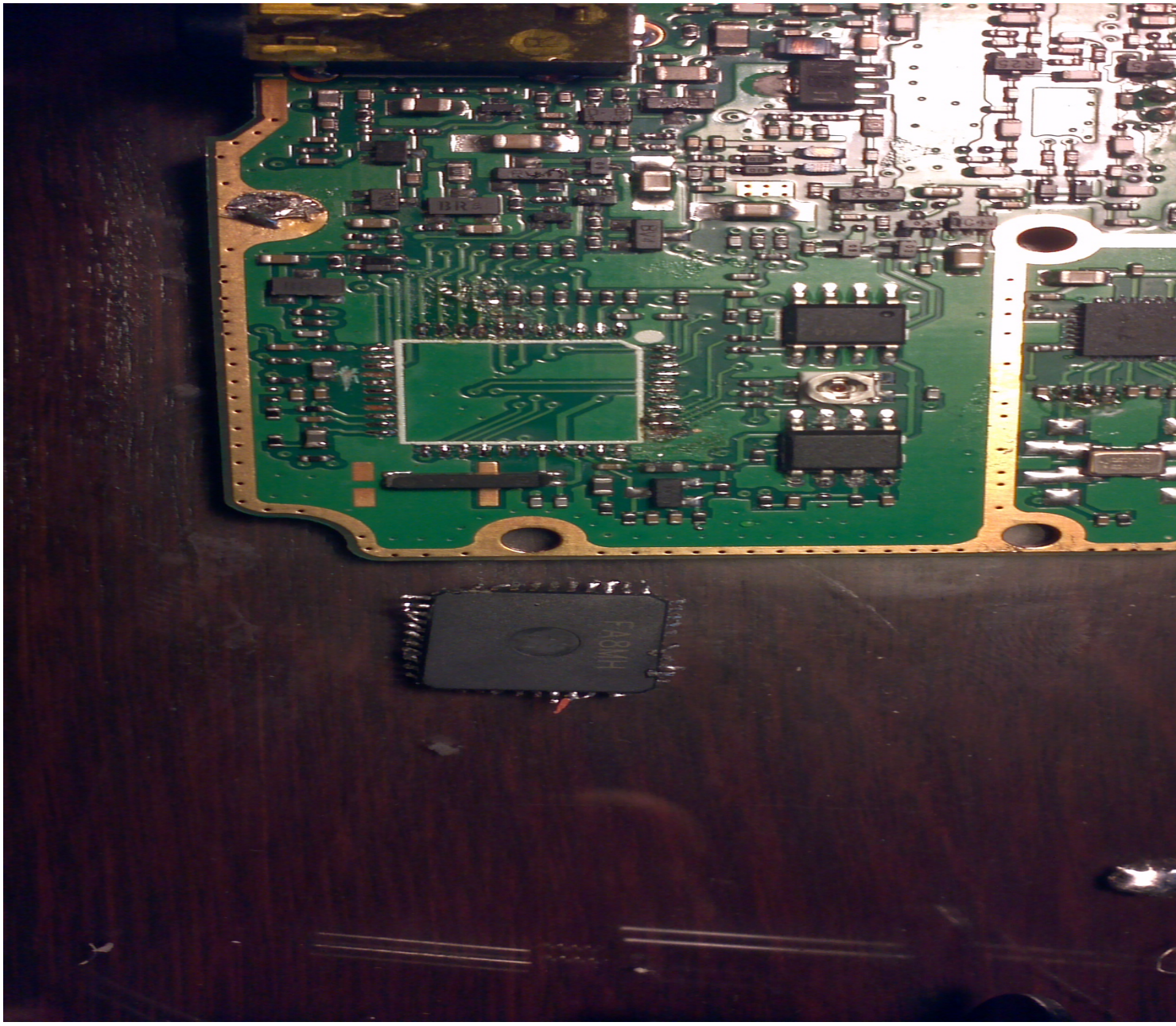
To set the TX/RX circuitry, here is what is needed: I did not actually TX anything, just measured the voltage on the TX power circuit.
There are 3 pins that need to be controlled: PC1 for RX power (3.3 to RX), P66 for TX power (0 to TX), and P56 for UHF/VHF mode (0 for UHF and 3.3 for VHF).

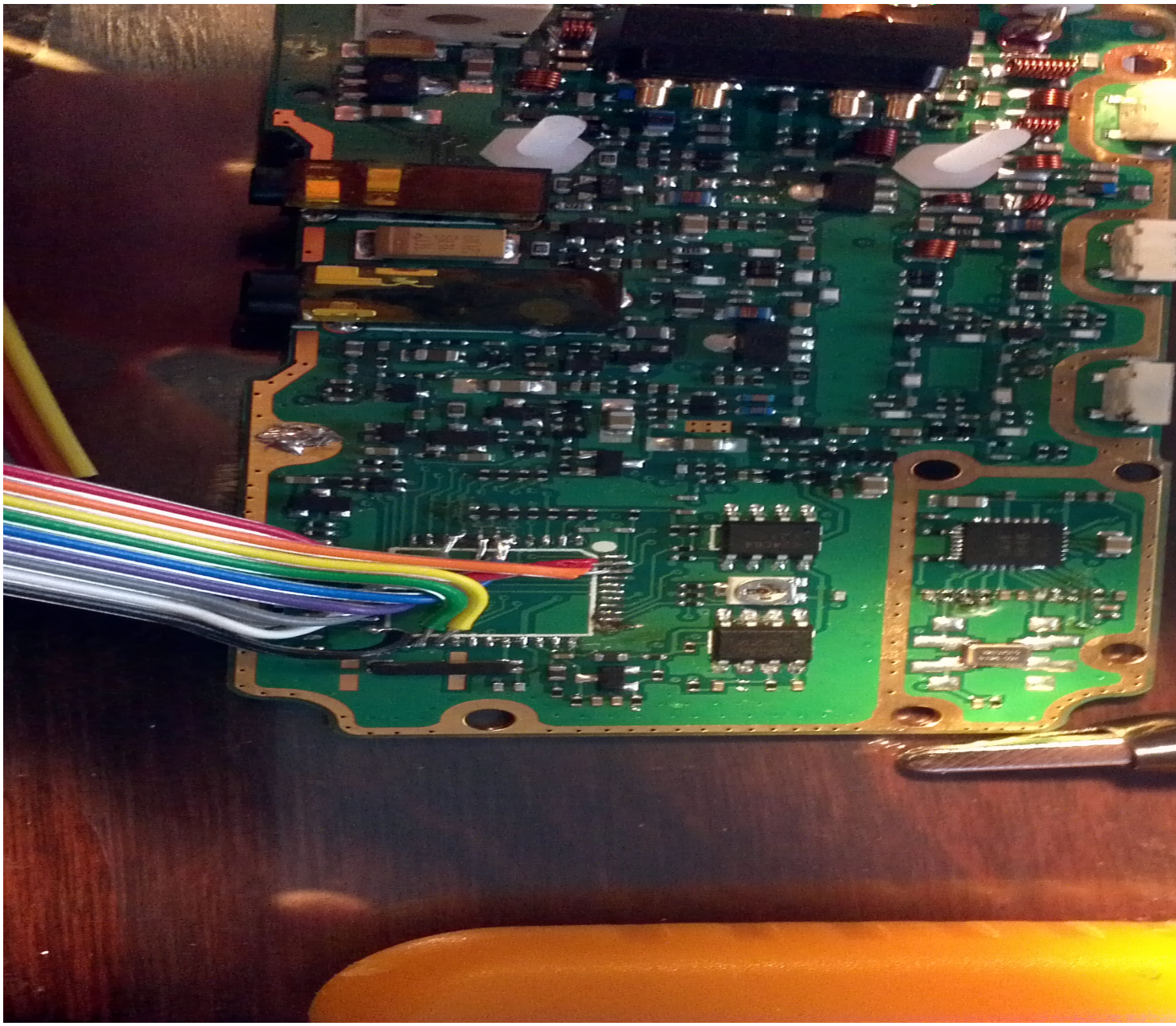
Here is a video of the radio receiving the NOAA channel
{youtube}6L9vMI4zKMM{/youtube}

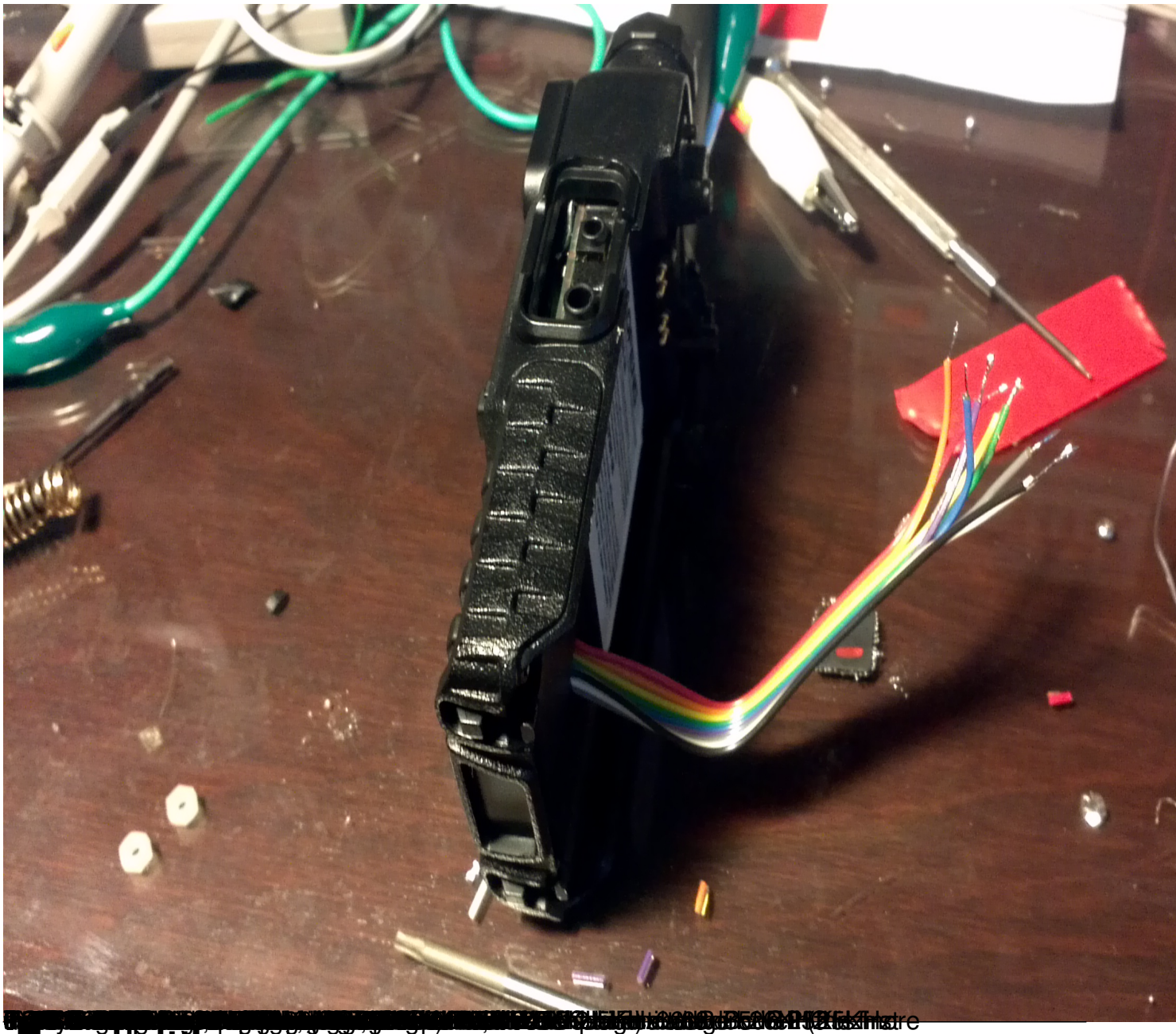
There is a bit of static, but that is what I get with a new radio as well. I can also set it to the police dispatch on UHF and it works as well.

# Removing the CPU

I took the plunge, removed the main MCU and just soldered wires to the pads (I did not have a rework station, so I used my soldering iron. I just ordered a rework station from sparkfun). Everything seems to work ok and I am able to fully configure the RDA chip and set auto sq, freq, etc.


F48MH





## Transmitting

I finally got my license a few weeks ago (KK6BWA) and after messing around a bit with just talking to people, I finally got back to the radio and tried to transmit. I also, tried to read some of the read only registers on the radio, which indicate RX level strength, DTMF decoded tones, etc.

The first thing I noticed was that I was able to tell the RDA1846 to send a sin wave of a given frequency. In fact, you can choose whether you want to send one sin wave or two at the same time (for DTMF). It also looks like you can change the sin wave relatively quickly  while transmitting, so I can even send 1200 Baud FSK signals. I wrote a simple morse code function to send my call sign (KK6BWA) though the radio for testing.

Here is a video of the radio sending the morse code over 145.525MHz.
{youtube}MSf7hUnp1_s{/youtube}

Since the chip itself can be configured to send data over the 220MHz band. I tried that as well. However, I only have another UV5R, so I have nothing to receive the signal on. Thankfully, Steve (WB8GRS) suggested to use harmonics and tune the other UV5R to the first harmonic. I configured the hacked radio to TX on 223.5MHz and the other UV5R to receive over (223.5*2) 447MHz. Since the RDA1846 has a special register that needs to be set to put it into the 220MHz band, I tried TXing with the register set to the other bands while on 220MHz. Unfortunately, by only setting the register to the 220MHz band, I was able to receive the signal, confirming the the stock UV5R will not be able to TX/RX on that band without setting this register. To my surprise I was able to receive the harmonics about 400 feet away before the signal started degrading.

Here is a video of the radio TXing on 223.5MHz and receiving over 447MHz.
{youtube}TWIO67yaZ8U{/youtube}

The RDA1846 is also able to set the TX deviation, so I tried messing around with that register to see if I can get a better transmission over the harmonics. Here is the same test as above, but each time I set the deviation param to a different number:
{youtube}d13mi6wl9Rk{/youtube}

Lastly, I tried to read the internal registers on the radio, which indicate the signal level strength, Voice signal strength, DTMF decoded output and some flags. Anyone knows why would the signal level strength change when different frequencies (DTMF) was going over the air? The signal level strength does seem to be working, since I was getting different values when I was tuning the radio to other far away stations.

{youtube}vgPwW9Nn9OY{/youtube}

Next step would be to design a PCB that I can place an atmega or an arm chip in the radio.

Here is the final code used for testing. It allows you to set registers by "Saa uu ll " where aa is the register address, uu is the upper byte to set and ll is the lower byte to set. "Raa" to read a register at address aa. "V" to display register values like signal strength, DTMF, etc. "T" to transmit my call sign in morse.
Using the arduino serial monitor its simple to just copy and paste settings. For example, to initialize the chip, copy the following into the arduino serial monitor (insure you are at 38400 baud rate).
{code}
S30 00 01
S30 00 04
S04 0F D0
S0B 1A 10
S2B 32 C8

S2C 19 64
S32 62 7C
S33 0A F2
S47 2C 2F
S4E 29 3A
S54 1D 4C
S56 06 52
S6E 06 2D
S70 10 29
S7F 00 01
S05 00 1F
S7F 00 00
S30 30 06
{/code}

Then to receive the NOAA channel on 162.550MHz send the following sequence
{code}
S30 30 06
S29 00 13
S2A d7 b0
S0F 6b e4
S48 00 88
S49 01 b3
S30 30 06
S30 30 2E
{/code}

Here is the arduino code
{code}
/*************************************************/
// Interfacing with the uvr5 RDA1846 using arduino //
// Written by Lior Elazary KK6BWA //
// 2013 //
/*************************************************/

#include <Wire.h>
// Arduino analog input 5 - I2C SCL
// Arduino analog input 4 - I2C SDA

//The RDA1846 chip address
#define ADDRESS B1110001

/*****************************************/
// Simple Arduino Morse Beacon //
// Written by Mark VandeWettering K6HX //
// Email: k6hx@arrl.net //

```
/*******************************************/

struct t_mtab { char c, pat; } ;

struct t_mtab morsetab[] = {
 {'.', 106},
 {',', 115},
 {'?', 76},
 {'/', 41},
 {'A', 6},
 {'B', 17},
 {'C', 21},
 {'D', 9},
 {'E', 2},
 {'F', 20},
 {'G', 11},
 {'H', 16},
 {'I', 4},
 {'J', 30},
 {'K', 13},
 {'L', 18},
 {'M', 7},
 {'N', 5},
 {'O', 15},
 {'P', 22},
 {'Q', 27},
 {'R', 10},
 {'S', 8},
 {'T', 3},
 {'U', 12},
 {'V', 24},
 {'W', 14},
 {'X', 25},
 {'Y', 29},
 {'Z', 19},
 {'1', 62},
 {'2', 60},
 {'3', 56},
 {'4', 48},
 {'5', 32},
 {'6', 33},
 {'7', 35},
 {'8', 39},
 {'9', 47},
 {'0', 63}
} ;
```

```
#define N_MORSE  (sizeof(morsetab)/sizeof(morsetab[0]))

#define SPEED  (20)
#define DOTLEN  (1200/SPEED)
#define DASHLEN  (3*(1200/SPEED))

void dash()
{
 delay(10);
 Wire.beginTransmission(ADDRESS);
 Wire.write(0x36);
 Wire.write(0x09);
 Wire.write(0x00);
 Wire.endTransmission(1);

 delay(DASHLEN);
 delay(10);
 Wire.beginTransmission(ADDRESS);
 Wire.write(0x36);
 Wire.write(0x00);
 Wire.write(0x00);
 Wire.endTransmission(1);

 delay(DOTLEN) ;
}

void dit()
{
 delay(10);
 Wire.beginTransmission(ADDRESS);
 Wire.write(0x36);
 Wire.write(0x09);
 Wire.write(0x00);
 Wire.endTransmission(1);

 delay(DOTLEN);
 delay(10);
 Wire.beginTransmission(ADDRESS);
 Wire.write(0x36);
 Wire.write(0x00);
 Wire.write(0x00);
 Wire.endTransmission(1);
 delay(DOTLEN);
}

void send(char c)
```

```
{
 int i ;
 if (c == ' ') {
 Serial.print(c) ;
 delay(7*DOTLEN) ;
 return ;
 }
 for (i=0; i<N_MORSE; i++) {
 if (morsetab[i].c == c) {
 unsigned char p = morsetab[i].pat ;
 Serial.print(morsetab[i].c) ;

 while (p != 1) {
 if (p & 1)
 dash() ;
 else
 dit() ;
 p = p / 2 ;
 }
 delay(2*DOTLEN) ;
 return ;
 }
 }
 /* if we drop off the end, then we send a space */
 Serial.print("?") ;
}

void sendmsg(char *str)
{
 while (*str)
 send(*str++) ;
 Serial.println("");
}
/****************** End of morse lib **********************/


void setup() {
 Wire.begin();
 Serial.begin(38400);
}

byte getVal(char c)
{
 if(c >= '0' && c <= '9')
 return (byte)(c - '0');
 else
```

```
 return (byte)(c-'A'+10);
}

short int readReg(unsigned char reg)
{
 Wire.beginTransmission(ADDRESS);
 Wire.write(reg);
 Wire.endTransmission(0);

 Wire.beginTransmission(ADDRESS);
 Wire.requestFrom(ADDRESS,2);
 unsigned char rDataU = Wire.read();
 unsigned char rDataL = Wire.read();
 Wire.endTransmission(1);
 short int data = word(rDataU, rDataL);
 return data;
}

void loop()
{
 // send data only when you receive data:
 if (Serial.available() > 0)
 {
 unsigned char d = Serial.read();
 if (d == 'S')
 {
 int i=0;
 char data[8];
 while(i < 8)
 if (Serial.available() > 0)
 data[i++] = Serial.read();

 unsigned char address = getVal(data[1]) + (getVal(data[0]) << 4);
 unsigned char dataU = getVal(data[4]) + (getVal(data[3]) << 4);
 unsigned char dataL = getVal(data[7]) + (getVal(data[6]) << 4);
 Serial.println(address, HEX);
 Serial.println(dataU, HEX);
 Serial.println(dataL, HEX);
 Serial.println();
 Wire.beginTransmission(ADDRESS);
 Wire.write(address);
 Wire.write(dataU);
 Wire.write(dataL);
 Wire.endTransmission(1);
 }
 if (d == 'R')
```

```
{
int i=0;
char data[2];
while(i < 2)
if (Serial.available() > 0)
data[i++] = Serial.read();

unsigned char address = getVal(data[1]) + (getVal(data[0]) << 4);
short int ss = readReg(address);
Serial.print("Register: ");
Serial.print(address, HEX);
Serial.print(" =");
Serial.println(ss, HEX);
}
else if (d == 'V')
{
//Read registers
short int ss = readReg(0x6C);
Serial.print("RX Signal Strength ");
Serial.println(ss);
short int vs = readReg(0x60);
Serial.print("Voice Signal Strength ");
Serial.println(vs);
short int dtmf = readReg(0x60);
Serial.print("DTMF ");
Serial.println(dtmf, HEX);
short int flags = readReg(0x5C);
Serial.print("Flags ");
Serial.println(flags, BIN);
delay(100);
Serial.write(27); // ESC
Serial.print("[2J"); // clear screen
Serial.write(27); // ESC
Serial.print("[H"); // cursor to home
}

if (d == 'T')
{
Wire.beginTransmission(ADDRESS);
Wire.write(0x30);
Wire.write(0x30);
Wire.write(0x46);
Wire.endTransmission(1);
sendmsg("KK6BWA ");
Wire.beginTransmission(ADDRESS);
Wire.write(0x30);
```

```
 Wire.write(0x30);
 Wire.write(0x06);
 Wire.endTransmission(1);
 }

 }
}
```

{/code}