EGRF-433A1

433MHz UHF ASK Data Transmitter and Receiver

Technical Manual
Rev 1r0

EGRF-433A1 transmitter and receiver pair is a low cost solution for radio wireless control application circuits. Operating at 433MHz ISM frequencies, this low power radio system has a remote operating reach of more than 100 meters in open field (125 meters/370 ft being typical) . The control distance can be stretched to 175meters, if short temporary disconnections due to RF signal drop outs can be tolerated.

Limitations

Before we go any further, it may be a good idea to know the EGRF-433A1 limitations. The most important thing to keep in mind is, the EGRF-433A1 is not a UART wireless cable replacement out of the box. It is, for a fact, a data transmitter-receiver system. But the thing is, its output is a raw, noisy, low level RF pulses that still requires a fair amount of signal processing before it becomes usable for data transmission.

The reception data, if present, will be available mixed with lots of random pulses (noise). The receiving MCU is expected to process the reception data to extract its content and examine the data if it is free from error and fit for use. Hence it transmits and receives data in a way not compatible with any UART protocol.

Programming an MCU to work with EGRF-433A1 does not sound so easy, and the fact is, it is really not. Arduino and gizDuino users are quite fortunate though. Someone already built a collection of routines to carry out these tasks, called the VirtualWire library. With the use of this library, using EGRF-433A1 with Arduino and gizDuino becomes as straightforward and painless even for beginners.

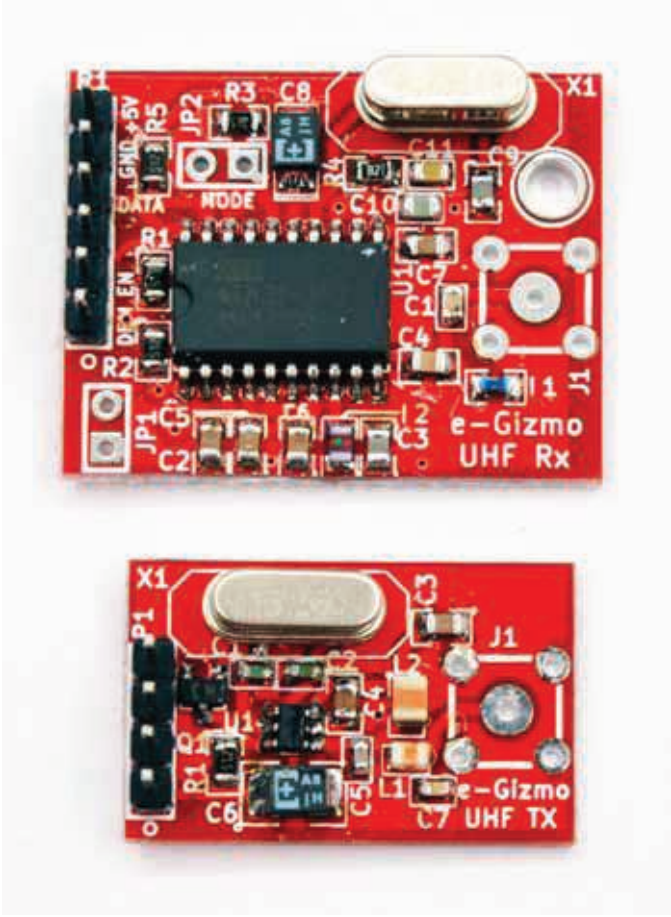

Figure 1. The EGRF-433A1 UHF Receiver (larger of the two) and Transmitter Module.

Figure 2. e-Gizmo Remote Controller with VirtualWire compatible protocol firmware fitted with EGRF-433A1-T transmitter module, ready for a gizDuino based wireless remote robot control application.

EGRF-433A1-T TRANSMITTER CIRCUIT

The transmitter module is based on Micrel's UHF transmitter MICRF113. This is a very stable UHF oscillator locked to a crystal frequency. It is capable of delivering RF output of up to 10dbm, although in our circuit, actual RF output is about 3 to 4dB less because of external components influence.

The transmitter is ASK pulse modulated by applying LVTTL level pulses at the TXIN input. TXIN input is non linear. Basically, what the input pulses does is turn ON and OFF the transmitter at a succession to effect ASK transmission.

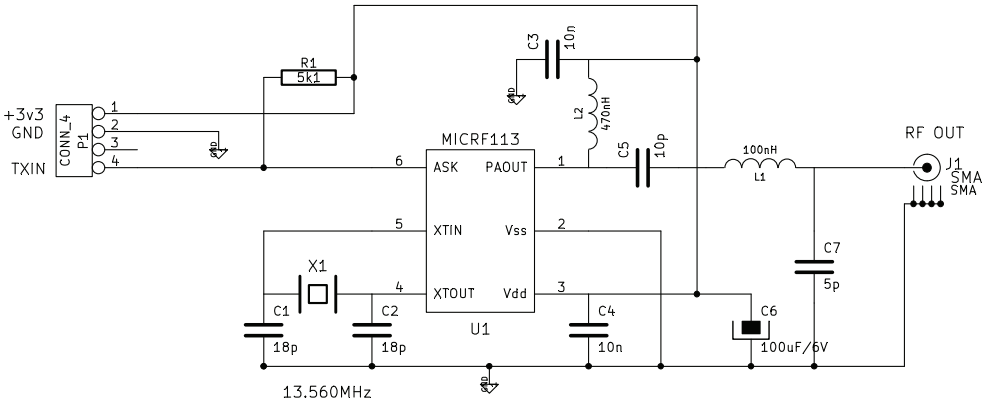

Figure 3. The transmitter module is crystal locked in frequency, ensuring a stable and more reliable transmitting function. All external components are fixed in value. Nothing can be accidentally detuned.

Table 1. Interface Port P1

PIN	ID	Description
1	+3v3	Power Input, 1v8 to 3v6 volts
2	GND	Power Ground
3		No connection
4	TXIN	ASK Data Pulse input

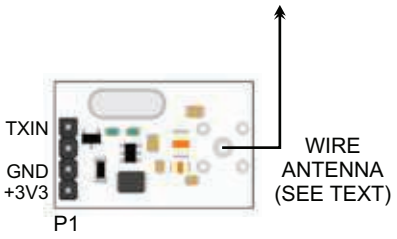

Figure 4. The transmitter module component layout. A stranded core insulated wire cut to 17cm in length can be used as an antenna.

EGRF-433A1-R RECEIVER CIRCUIT

The receiver module is built around ATA3741, a UHF receiver circuit capable of demodulating either an ASK or FSK signal. The receiver frequency is likewise crystal locked to keep the frequency from wandering off and getting out of tune with the transmitter.

The output of the ATA3741 is already in the form of digital pulses. But as it is, it is not ready for use. As I already mentioned, we could expect a data output mixed with all sorts of noise. Hence, to ensure succesful data transfer, the receiver hosts MCU need something it could easily recognize from noise to get it start processing of reception data and deal with the separation of data from noise more effectively. For this, the data packet is send by the transmitting device with extra payloads for synchronization and error detection. The payload depends on the protocol used. Following is the scary detail.

VirtualWire starts transmission by sending a series of synchronization pulses, 18 pairs of 1 and 0s to be exact. The hosts MCU running the VirtualWire reception routine continuously checks for this sync pulses, and will only start processing upon the detection of sync signal. The sync pulses are immediately followed by a start marker - a 12bit pulse pattern, the number of bytes to transmit, and the data stream itself. Transmission is concluded with a 4 byte CRC that must be used by the host controller to validate the data.

And we are not finished yet. All data after the start marker are encoded in 4b/6b format in order not the upset the DC bias of the ATA3741 receiver PLL circuit. Any undesired DC bias within the PLL control loop will almost guarantee a data lost.

Fortunately for Arduino and gizDuino programmers, you need not worry about these details. Every complicated thing just mentioned had already been taken cared of by the VirtualWire library, so that you don't have to. It is a free Arduino IDE add-on library that can work with EGRF-433A1 wireless hardware. If you haven't done so, now is the good time to download this library.

Users of non-Arduino compatible platform/MCUs are not as lucky, and may have to do the dirty work themselves. For now, Google is your best friend.

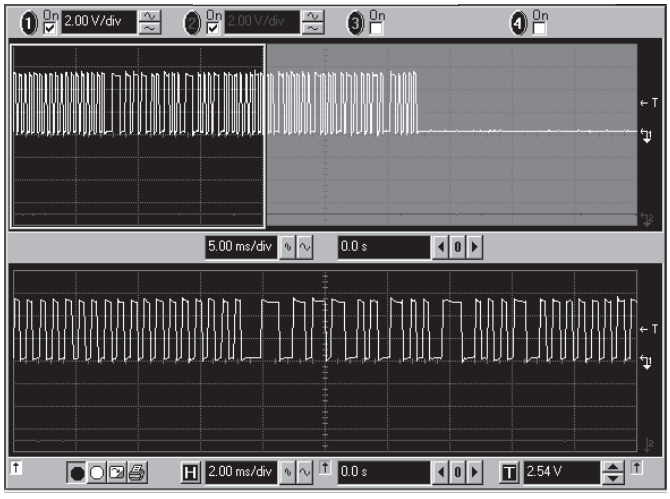

Figure 5. A snapshot of an oscilloscope trace of a EGRF-433A1-T transmitter module sending the message "123456". The lower trace is the expanded highlighted portion of the packet. The lower trace clearly shows the synchronization pulses, the 12-bit start marker, encoded payload specifier, and portions of the payload itself.

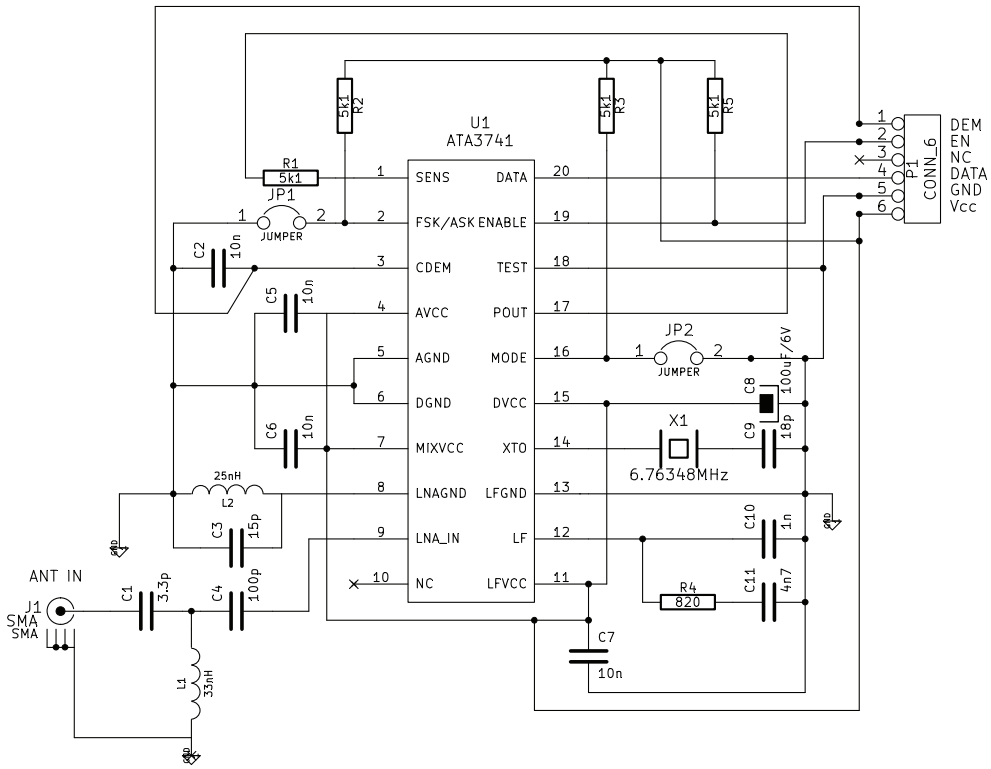

Figure 6. EGRF-433A1-R Receiver Module schematic. Like its transmitter pair, the circuit uses fixed valued components even on RF section. This is a PLL receiver circuit frequency locked to a crystal reference. It is configured for ASK reception, but users can configure it in FSK mode by installing JP1. Note that FSK will not with EGRF-433A1-T transmitter module.

Table 2. P1 Interface Port

PIN	ID	DESCRIPTION
1	DEM	Analog Demodulator Output
2	EN	Enable Input, Normally open (“1”)
3	N.C.	No connection
4	DATA	Data Output
5	GND	Common Ground
6	Vcc	+5V Power Input

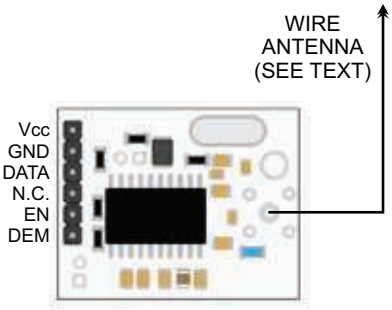

Figure 7. EGRF-433A1-R Receiver Module components layout. The antenna can likewise be fashioned out of a 17cm length of stranded insulated wire.

APPLICATION EXAMPLES

To use the EGRF-433A1 with your gizDuino, you need to integrate the VirtualWire library with your current Arduino IDE installation. Installing Virtual-Wire Libraries to your Arduino IDE requires only a couple of steps:

1. Download the latest VirtualWire library from http://www.open.com.au/mikem/arduino/

2. Unzip the file and copy the whole VirtualWire folder to your Arduino IDE directory libraries sub-folder (e.g. E:\arduino-1.0.1\libraries)

This library will run with all gizDuino platform, including the minis.

Transmitter Application Example

The transmitter circuit requires just one I/O line driving the TXIN. Any DIO can be used to drive the TXIN. But it is probably a good idea to keep the UART port pin 0, pin 1, and pin 2 of the gizDuino free. The UART port is one of the most useful features of a microcontroller. A lot of devices would want to connect in the UART port. And it is not needed for EGRF-433A1 to work, although you can use it with equal ease if you want to. It will just be a waste. You will never know when you will need it for other devices.

In the following example, it is connected to pin 14. An Arduino sample sketch for the transmitter is shown in the following page.

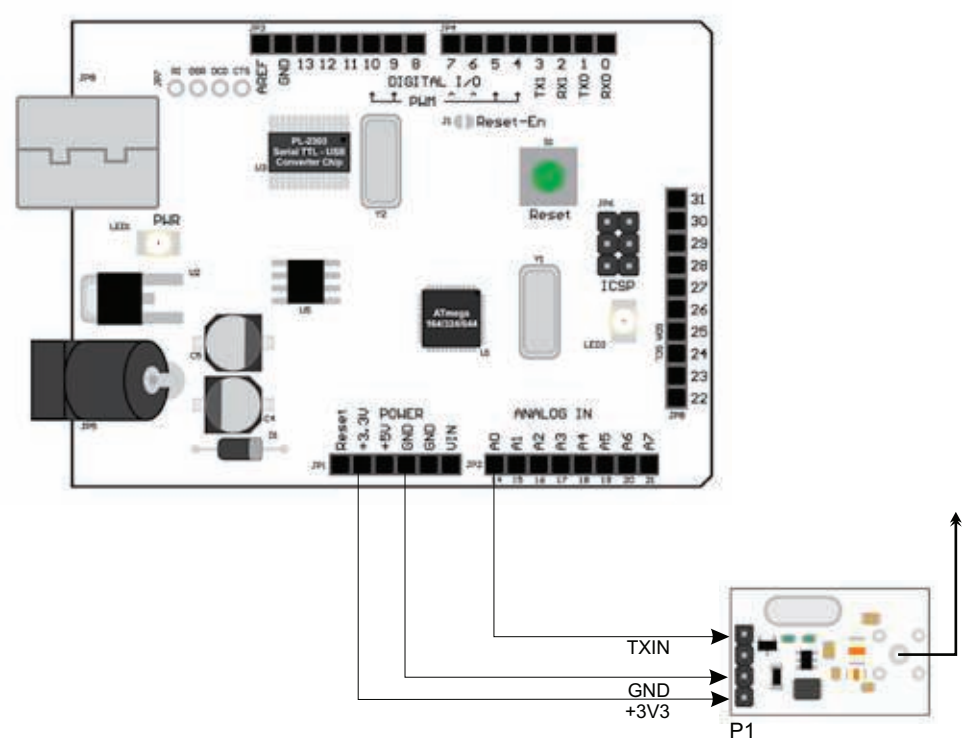

Figure 8. Example of an EGRF-433A1-T transmitter module wired with gizDuino+ module. Needles to say, this setup will work with any Arduino and compatible module as well, not just with the gizDuino+. User can freely use any other DIO pin other than the one shown in this example.

```
/* Sample Arduino Sketch for
*  e-Gizmo EGRF-433A1-T
*  UHF Transmitter Module
*  Using VirtualWire Protocol
*
*  This will work with all gizDuino and Arduino platform
*/

#include <VirtualWire.h>


// DIO txpin assignment. Change pin assignment if you want to
const  byte  txpin=14;  // tx pin assigned to DIO 14


void setup()
{

    vw_setup(4800);        // Transmission speed at 4800bps
    vw_set_tx_pin(txpin);   //tell VirtualWire which pin to use
    pinMode(txpin,OUTPUT);  //Configure this pin as an output. VirtualWire will not do it for you

}

void loop()
{

  char *msg=”123456”;  // test message “123456”

  // Send message via VirtualWire
   vw_send((uint8_t *)msg, strlen(msg));
  // That’s it!
    delay(300);
  }
```

Receiver Application Example

The receiver circuit, as a minimum, requires just one gizDuino DIO to implement. The host controller (gizDuino) running the VirtualWire services periodically checks the specified DIO. If an incoming stream is detected, it processes the data, get rid of the noise, checks for data integrity, and then expose the clean receive data to a running user codes. All complex tasks are done by the VirtualWire for you.

Control to the receiver ENable pin is not necessary, but is highly recommended. The EGRF-433A1 DATA output pin, as repeatedly mentioned, is littered with random noise pulses. Even with the absence of incoming signal, the DATA output never cease going from state to state (Fig. 11). This will conceivably put a burden in the MCU hosts, with the effect showing as system slowing down. The Enable pin helps minimize this. It works by keeping the DATA output silent (no pulses) when nothing is being sent from the transmitter side (Fig. 10). This is possible because the ATA3741 chip used in the receiver module has a built-in function that can detect the synchronization start pulse on its own. To use it, the host MCU must pull down Enable pin for a short moment after the completion of a data packet reception.

An Arduino demo sketch for the receiver module is shown in the following pages.

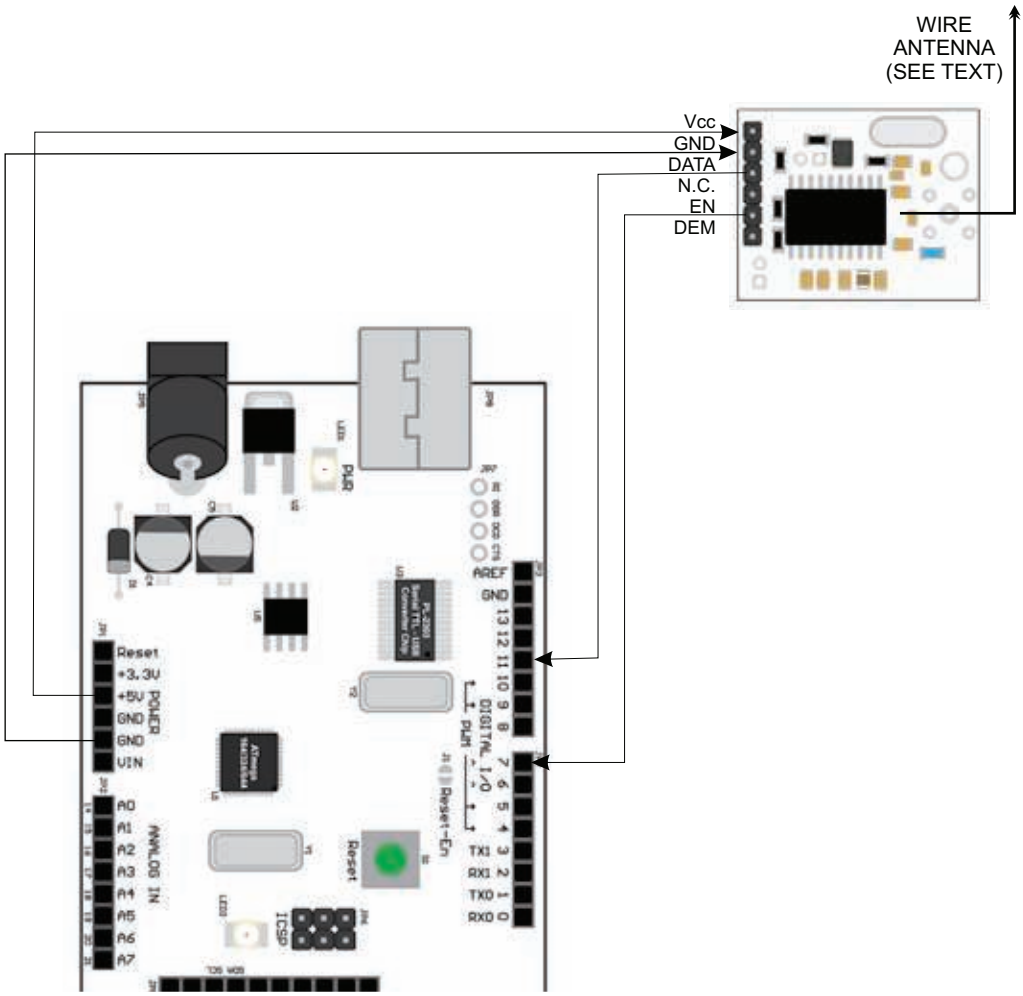

Figure 9. EGRF-433A1-R Receiver Module on a gizDuino+. As in the transmitter wiring example, users can freely choose a different DIO if the application circuit so requires. But remember to modify the pin assignment on your Arduino Sketch accordingly.

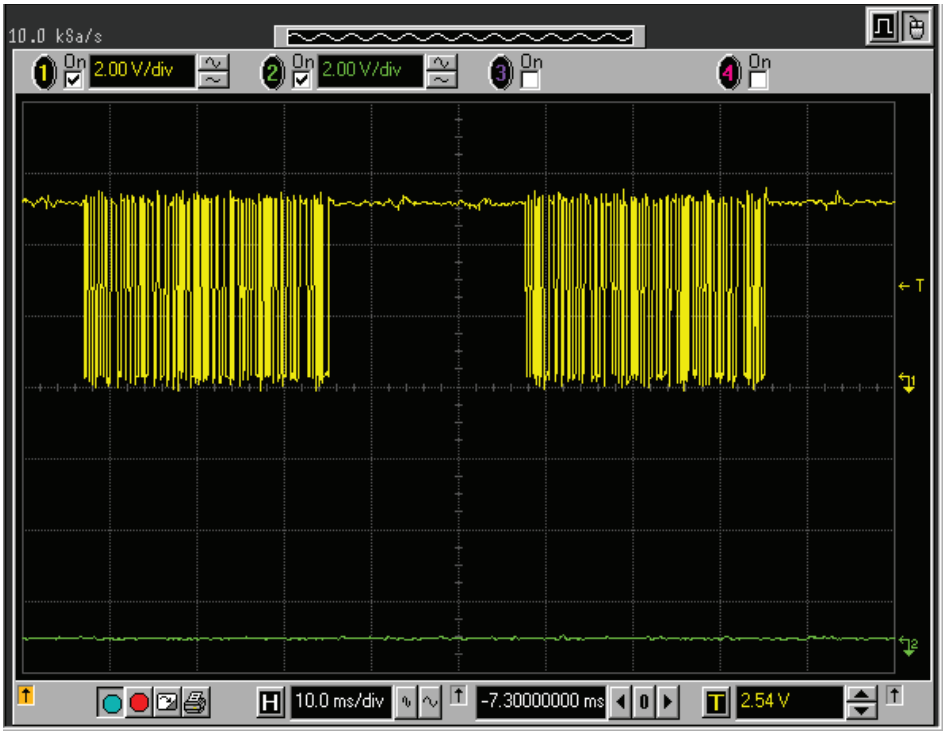

Figure 10. Oscilloscope trace of receiver DATA output with Enable pin in use. The Enable function mutes the DATA output after completing reception of a data packet, and then restart DATA output by itself upon detection of an incoming transmission.

Figure 11. You can leave out the Enable function. But this is what will happen in the DATA output. Random noise pulses keeps coming out of the DATA pin even in the absence of an incoming signal. This may present a unnecessary burden that could slow down the processing speed of the hosts MCU.

```
/*  Demo Rx program for e-Gizmo EGRF-433A1-R UHF Rx module *
*   by e-Gizmo Mechatronix Central
*
*   using VirtualWire Libraries
*
*  This sample program dumps Rxed data to serial port.
*  Monitor received data by using the IDE Serial Monitor
*/

#include <VirtualWire.h>

const  byte  rxpin=11;    // DIO pin 2 for Rx
const  byte  rxenable=7;  //DIO pin 7 for Rx Enable
const  byte  ledpin=13;    // Built in LED

/* EGRF-433A1-R Speed settings */

// select 9600bps or 4800bps
// Make sure only one group is uncommented

// Uncomment next two lines if you want to set speed to 9600baud
//unsigned int  spattern=0x7810;  //BR_Range3
//unsigned int  lpattern=0x2ee0;  // bit check limit 46-56

//Uncomment next two lines for 4800 baud
unsigned int  spattern=0x6810;  //BR_Range2
unsigned int  lpattern=0x2FD8;  // bit check limit for 4500-5100 bps

byte bitctr=14;

void setup()
{
   pinMode(rxenable,OUTPUT);    // EGRF-433A1 DIO for ENable pin
   ATA3741();                   // setup Rx module ATA3741 chip

   // UART is used for this demo to allow visual display
   // of Rxed data via Arduino IDE Tools>Serial Monitor
   // Otherwise, it is freely available for other purpose
  Serial.begin(9600);

   // Apply VitualWire settings
   vw_setup(4800);      // Bits per sec
   vw_set_rx_pin(rxpin);
   vw_rx_start();       // Start the receiver PLL running

   pinMode(ledpin,OUTPUT);  // LED
}
```

```
void loop()
{

   uint8_t buf[VW_MAX_MESSAGE_LEN];
   uint8_t buflen = VW_MAX_MESSAGE_LEN;
   int i;


    if (vw_get_message(buf, &buflen)){  // data available?

       digitalWrite(13, HIGH);         // flash ON LED

       // And dump Rxed data to serial port
       // Rxed data are stored in buf[] memory

          Serial.print("Rx Data: ");

          for (i = 0; i < buflen; i++)
           {
              Serial.print((char)buf[i]);
           }
          Serial.println("");

       digitalWrite(rxenable, LOW);  // Reset EGRF-433A1-R
       digitalWrite(13, LOW);        // Flash OFF LED indicator
       digitalWrite(rxenable, HIGH);  //Re enable to receive next data packet

  }
}

/* EGRF-433A1-R Hardware Initialization Routines
*  Configure the ATA3741 chip
*  to start RX activity
*/

void ATA3741(void){
  digitalWrite(rxenable,HIGH);

  // Initiate ATA3741 configuration mode by
  // forcing DATA pin low for 15ms
  pinMode(rxpin,OUTPUT);      // Change DATA pin direction to OUTPUT
  digitalWrite(rxpin,LOW);
  delayMicroseconds(15000);
  digitalWrite(rxpin,HIGH);

  pinMode(rxpin,INPUT);
  digitalWrite(rxpin,LOW);
  // wait for t2

  bitctr=14;
  // send 14 bits init pattern
  while(bitctr>0){
```

```
    while(digitalRead(rxpin)==HIGH);
    if((spattern & 0x8000)==0)
      ATA3741_LO();
      else
      ATA3741_HI();

     bitctr--;
     spattern=spattern<<1;
   }

   delayMicroseconds(15000);

   // Put back ATA3741 in config mode

   pinMode(rxpin,OUTPUT);
   digitalWrite(rxpin,LOW);
   delayMicroseconds(1000);
   digitalWrite(rxpin,HIGH);
   pinMode(rxpin,INPUT);
   digitalWrite(rxpin,LOW);

   // Enter 14-bit bit check limits pattern
    bitctr=14;
    while(bitctr>0){
    while(digitalRead(rxpin)==HIGH);
    if((lpattern & 0x8000)==0)
      ATA3741_LO();
      else
      ATA3741_HI();

     bitctr--;
     lpattern=lpattern<<1;
   }
}

// Output a logic low
void ATA3741_LO(void){
  // wait until rxpin goes hi
  while(digitalRead(rxpin)==LOW);

  delayMicroseconds(131);
  pinMode(rxpin,OUTPUT);    // output a low
  delayMicroseconds(150);    // for 150uS
  pinMode(rxpin,INPUT);
  delayMicroseconds(50);
}

// Output a logic high
void ATA3741_HI(void){
  // Since rxpin should be normally at logic high
  // just wait if necessary until rxpin goes high
  while(digitalRead(rxpin)==LOW);
}
```

PCB ARTWORK

Figure 12. EGRF-433A1-T Transmitter Module PCB artwork.

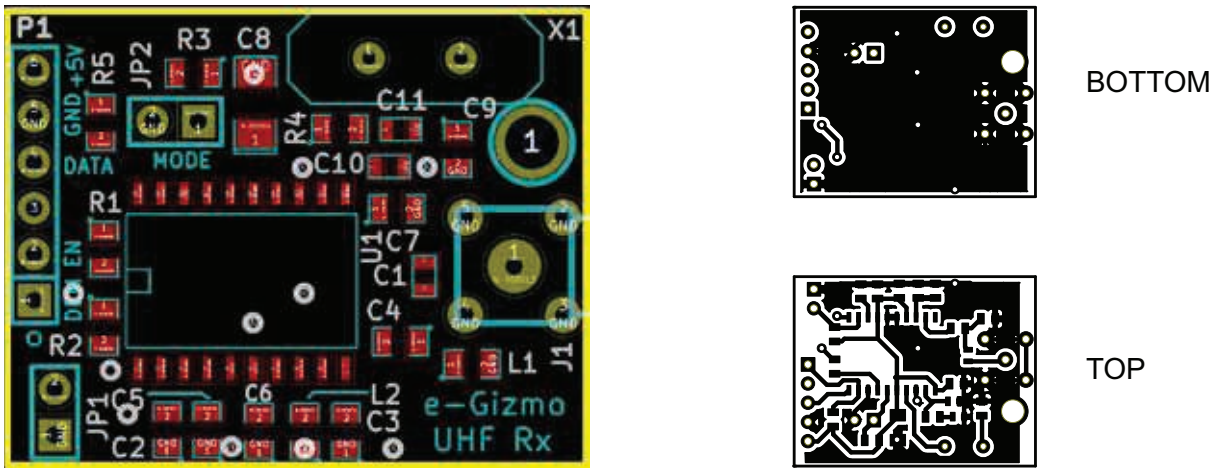

Figure 12. EGRF-433A1-R Receiver Module PCB artwork.