



You are here

[Home](#)

Running TX433 and RX433 RF modules with AVR microcontrollers

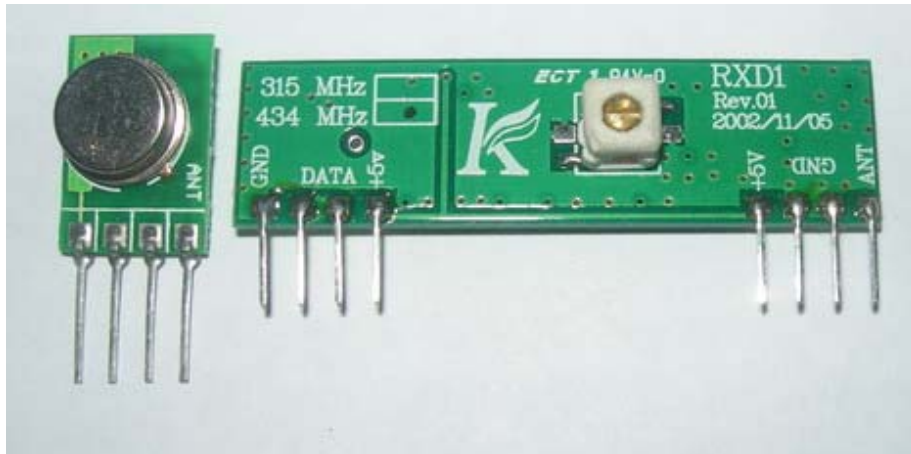
Fri, 03/14/2008 - 16:17 — admin

Topics:

[-Example AVR Projects](#)

Sometimes in embedded design you may want to go wireless. Might be you will want to log various readings of remotely placed sensors, or simply build a remote control for robot or car alarm system.

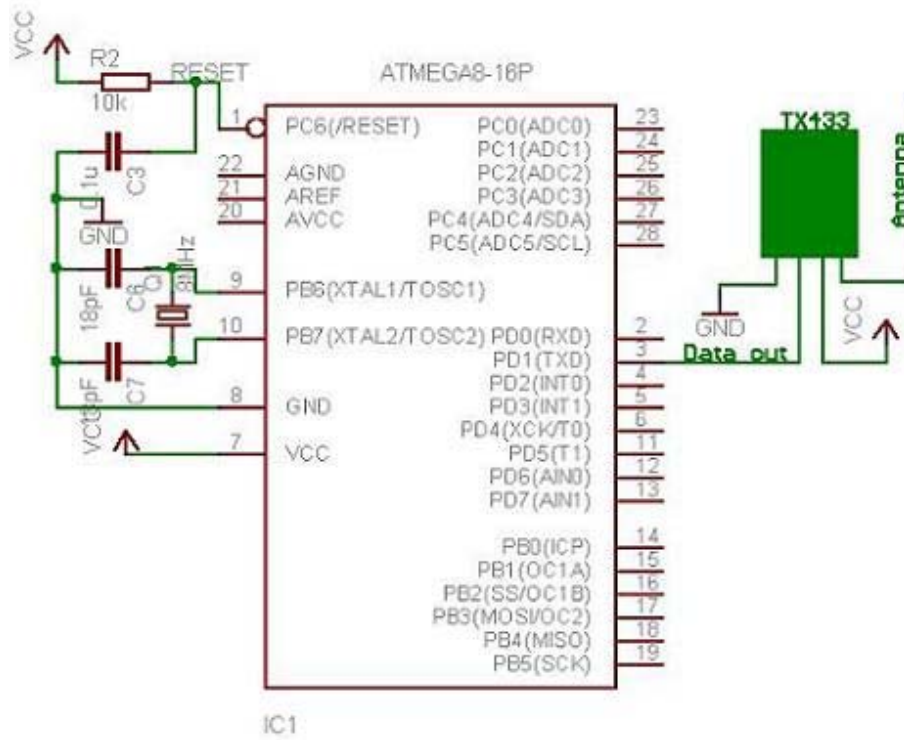
Radio communications between two AVR microcontrollers can be easy when specialized modules are used. Lets try to run very well known RF modules [TX433](#) and [RX433](#) that (or similar) can be found almost in every electronics shop and pair of them cost about ~15 bucks.



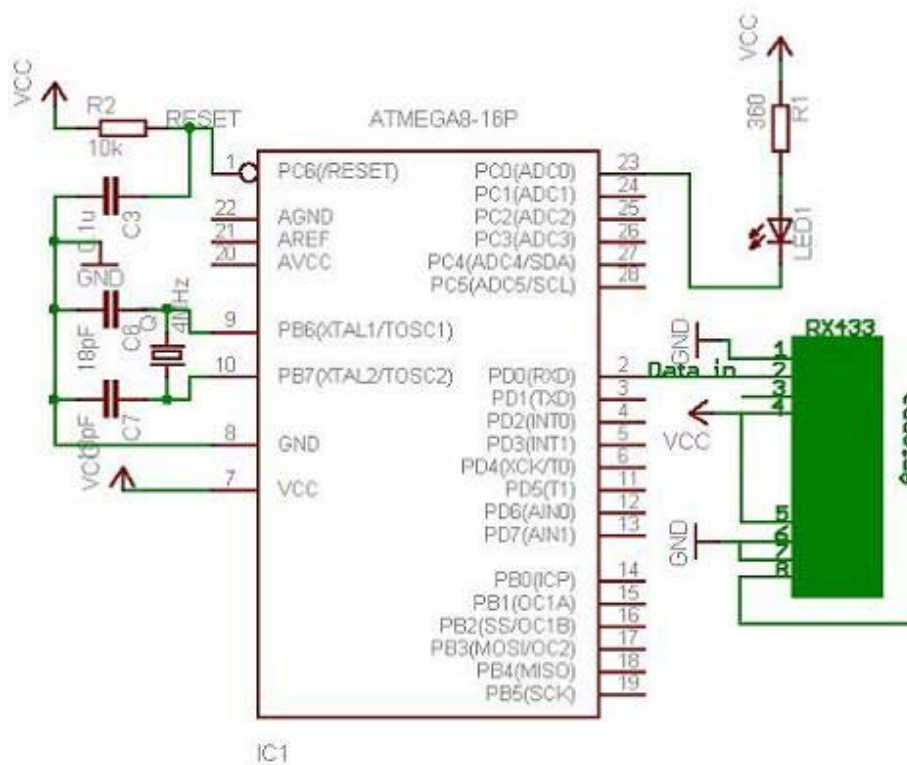
Transmitter and receiver modules are tuned to work correctly at 433.92MHz. Transmitter can be powered from 3 to 12V power supply while receiver accepts 5V. 5V is common for AVR microcontrollers so no problems with interfacing. Modules don't require addition components – just apply power and connect single data line to send information to/from and that's it. For better distances apply 30 – 35cm antennas. Modules use Amplitude-Shift Keying([ASK](#)) modulation method and uses 1MHz bandwidth.

I have constructed two separate circuits for testing on Atmega8 microcontrollers.

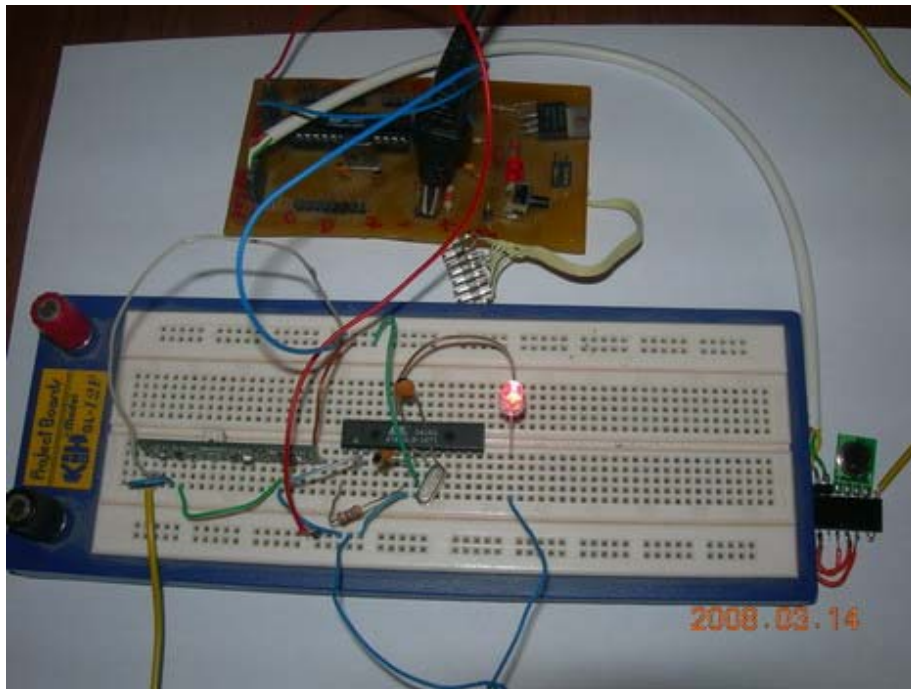
Transmitter



Receiver



For testing I have used a prototyping board and breadboard.



As you can see I have used one LED for indicating RF activity. Ok enough about hardware part – actually there is nothing more to say – circuits are simple.

Lets move on to software part. Radio transmission is a bit more complicated than wired communications because you never know what radio signals are present on air. So all matters how transmitted signal is encoded. And this is a part where you have many choices: use hardware encoding like USART or write your own based on one of many ending methods like NRZ, Manchester etc. In my example I have used AVR USART module to form data packs. Using hardware encoders solves many problems like synchronization, start and stop, various signal checks. But as long as I was practising you cannot rely on plain USART signal. Here you can actually improvize by adding various checks and so on.

I decided to form 4 byte data packages in order to send one byte information. These include:

- one **dummy** synchronization byte (10101010);
- one **address** byte – in case there are more receivers(or transmitters);
- one **data** byte;
- and **checksum** which is actually a sum of address and data(address+data).

Why did I use a dummy byte at the beginning of package. Simply I noticed, that when transmitter doesn't transmit any data – receiver catches various noises that come from power supply or other sources because receiver likes adjust its input gain depending on input signal level. First byte tunes receiver to accept normal signal after then address byte, data and checksum can be read more reliably. Probably with different transmission modules you may exclude this dummy byte.

Transmitter program for AVR Atmega8:

```
#include <avr/io.h>

#include <util/delay.h>

#ifndef F_CPU

//define cpu clock speed if not defined

#define F_CPU 8000000
```

```

#endif

//set desired baud rate

#define BAUDRATE 1200

//calculate UBRR value

#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)

//define receive parameters

#define SYNC 0XAA// synchro signal

#define RADDR 0x44

#define LEDON 0x11//switch led on command

#define LEDOFF 0x22//switch led off command

void USART_Init(void)
{
    //Set baud rate

    UBRRL=(uint8_t)UBRRVAL;          //low byte

    UBRRH=(UBRRVAL>>8);          //high byte

    //Set data frame format: asynchronous mode,no parity, 1 stop bit, 8 bit size
    UCSRC=(1<<URSEL)|(0<<UMSEL)|(0<<UPM1)|(0<<UPM0)|
           (0<<USBS)|(0<<UCSZ2)|(1<<UCSZ1)|(1<<UCSZ0);

    //Enable Transmitter and Receiver and Interrupt on receive complete

    UCSRB=(1<<TXEN);
}

void USART_vSendByte(uint8_t u8Data)
{
    // Wait if a byte is being transmitted

    while((UCSRA&(1<<UDRE)) == 0);

    // Transmit data

    UDR = u8Data;
}

void Send_Packet(uint8_t addr, uint8_t cmd)
{
    USART_vSendByte(SYNC);//send synchro byte

    USART_vSendByte(addr);//send receiver address

    USART_vSendByte(cmd);//send increment command

    USART_vSendByte((addr+cmd));//send checksum
}

void delaysms(uint8_t t)//delay in ms
{

```

```

uint8_t i;

for(i=0;i<t;i++)

    _delay_ms(1);

}

int main(void)

{

USART_Init();

while(1)

    {
    //endless transmission

    //send command to switch led ON

    Send_Packet(RADDR, LEDON);

    delayms(100);

    //send command to switch led ON

    Send_Packet(RADDR, LEDOFF);

    delayms(100);

    }

    return 0;

}

```

In my case I used UART 1200 baud rate. It may be increased or decreased depending on distance and environment. For longer distances lower baud rates works better as there is bigger probability for transmission errors. Maximum bit rate of transmitter is 8kbits/s what is about 2400 baud. But what works in theory usually do not work in practice. So 1200 baud is maximum what I could get working correctly.

Transmitter sends two commands (LEDON and LEDOFF) to receiver with 100ms gaps. Receiver recognizes these commands and switches LED on or off depending on received command. This way I can monitor if data transfer works correctly. If LED blink is periodical – then transmission goes without errors. If there is an error in received data then LED gives shorter blink.

Receiver program code:

```

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#ifndef F_CPU

//define cpu clock speed if not defined

#define F_CPU 4000000

#endif

//set desired baud rate

#define BAUDRATE 1200

//calculate UBRR value

#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)

```

```

//define receive parameters

#define SYNC 0XAA// synchro signal

#define RADDR 0x44

#define LEDON 0x11//LED on command

#define LEDOFF 0x22//LED off command

void USART_Init(void)

{

    //Set baud rate

    UBRRL=(uint8_t)UBRRVAL;          //low byte

    UBRRH=(UBRRVAL>>8);          //high byte

    //Set data frame format: asynchronous mode,no parity, 1 stop bit, 8 bit size

    UCSRC=(1<<URSEL)|(0<<UMSEL)|(0<<UPM1)|(0<<UPM0)|

            (0<<USBS)|(0<<UCSZ2)|(1<<UCSZ1)|(1<<UCSZ0);

    //Enable Transmitter and Receiver and Interrupt on receive complete

    UCSRB=(1<<RXEN)|(1<<RXCIE); //|(1<<TXEN);

    //enable global interrupts

}

uint8_t USART_vReceiveByte(void)

{

    // Wait until a byte has been received

    while((UCSRA&(1<<RXC)) == 0);

    // Return received data

    return UDR;

}

ISR(USART_RXC_vect)

{

    //define variables

    uint8_t raddress, data, chk;//transmitter address

    //receive destination address

    raddress=USART_vReceiveByte();

    //receive data

    data=USART_vReceiveByte();

    //receive checksum

    chk=USART_vReceiveByte();

    //compare received checksum with calculated

    if(chk==(raddress+data))//if match perform operations

    {

```

```

        //if transmitter address match
        if(raddress==RADDR)
        {
            if(data==LEDON)
            {
                PORTC&=~(1<<0); //LED ON
            }
            else if(data==LEDOFF)
            {
                PORTC|=(1<<0); //LED OFF
            }
            else
            {
                //blink led as error
                PORTC|=(1<<0); //LED OFF
                _delay_ms(10);
                PORTC&=~(1<<0); //LED ON
            }
        }
    }
}

void Main_Init(void)
{
    PORTC|=(1<<0); //LED OFF
    DDRC=0X001; //define port C pin 0 as output;
    //enable global interrupts
    sei();
}

int main(void)
{
    Main_Init();
    USART_Init();
    while(1)
    {
    }
    //nothing here interrupts are working
    return 0;
}

```

}

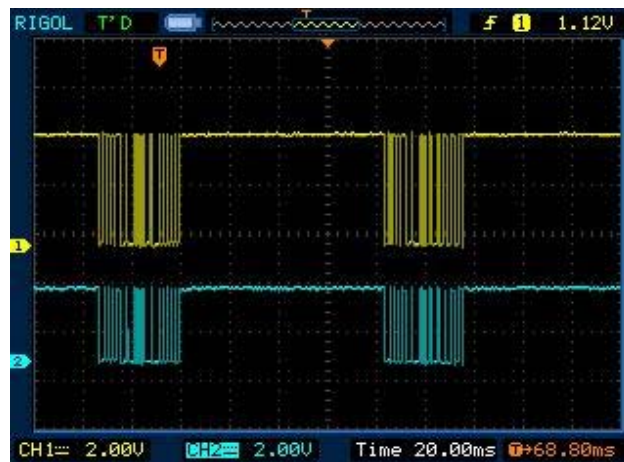
Receiver program receives all four bytes, then checks if checksum of received bytes is same as received checksum value. If checksum test passes then receiver addresses are compared and if signal is addressed to receiver it analyses data.

After all I have noticed that without antennas transmission is more erroneous even if modules are standing near by. Of course with all my power chords around the room I was getting lots of noises that receiver was catching between data transmissions.

In the last pictures you can see data packets of 4 bytes seen on the oscilloscopes. Yellow signal is from transmission data line(TX) while blue is taken from receiver data line(RX):



Transmitted and received signals matches



There is no noise between data packages(ideal case)

Good luck if you will decide to go wireless.

Receiver and transmitter WinAVR program codes for ATmega8 are [here](#) .

- [Add new comment](#)

Comments

[About dummy byte](#)

admin - Fri, 03/14/2008 - 17:21.

If you have any questions or suggestions feel free to comment.

- [reply](#)

Modular programming

mr23 - Tue, 03/25/2008 - 10:54.

Please write a simple article about how to do modular programming with WinAVR + AVRstudio in C. With multiple .C source files, headers, global variables, prototypes, common #define-s. I couldn't find any useful resources on the net about this. Thank you!

- [reply](#)

Modular programming

admin - Tue, 03/25/2008 - 17:49.

I will consider this. Thank you for suggestion.

- [reply](#)

Help with a TX433

bfrye - Tue, 06/17/2008 - 03:56.

I just bought a pair of the RX/TX 433 pair. I am trying to get the things running... My very first step was to plug in the receiver and look at the data line. It is all over the place! Then I hooked up the transmitter and just hard wired the data line to +5V. I expected to see the receiver signal doing something reasonable. It seems like no matter what I am doing I have garbage on the data line. I have an antenna and everything... Any input? Brandon

- [reply](#)

tx/rx433

bfrye - Tue, 06/17/2008 - 17:07.

(Hopefully this is not a re-post). I am trying to use these modules in my wireless application. I am having a lot of trouble. I seem to be getting nothing but noise on the data pin... 30 cm antenna, modules are several feet apart, proper grounding etc... Any ideas? When I put the scope on the data line of the receiver, I get garbage. The transmitting/powered down/not transmitting seems to make little difference. Am I missing something?

- [reply](#)

Rx Tx \$22Mhz Atmega128

naveen89 - Sun, 02/21/2010 - 06:00.

Sir, I have Atmega128. I am trying to run your code on my Microcontroller but am not able to get any output. I modified code for Atmega128, Since your code was for Atmega8. `F_CPU to 16000000` `BAUDRATE 19200` `For TxUBRR1L=0x67; //Lower UBRR1H=0x00; //Higher bits for UBRR=103 UCSR1A=0x00; UCSR1B=0x18; //Enable Receiver and Transmitter UCSR1C=0x06; For RxUBRR1L=0x67; //Lower UBRR1H=0x00; //Higher bits for UBRR=103 UCSR1A=0x00; UCSR1B=0x18; //Enable Receiver and Transmitter UCSR1C=0x06; But still I am not getting any output. Please help. Regards`

- [reply](#)

Non-inverted TXD line.

kazkowicz - Mon, 05/03/2010 - 15:33.

Idle TXD line is held high. IMHO this will cause continuous carrier transmission when no data is being sent. TXD signal should be inverted before feeding to the transmitter and inverted once again after receiver to restore correct logic levels. Simple transistor inverters will do the job. If for some reason receiver is connected to PC COM port, the second inversion is not necessary.

- [reply](#)

It's your baud rate

NOVAfarmer - Sun, 07/25/2010 - 15:28.

I didn't look through your whole code, but the first problem I saw was that you are trying to use a baud rate of 19200. These radio modules can only be used at 2400 max.

- [reply](#)

adding different remotes?

sherazil1 - Sun, 08/15/2010 - 09:36.

i am starting a fixed code remote project ... and i am really thankful to you for this great tutorial.. but i have some queries? and that is that i want the receiver to be locked to a number of transmitters... so how i am going to do it...should i program each transmitter with different address? and place a push button if pressed the receiver takes the transmitted signal and stores the address in its eeprom.. and afterwards it matches each signal received with that codes and makes a valid or invalid decision...can u help me to make a professional approach regards

- [reply](#)

No output from the receiver

piyushpandey - Sun, 04/03/2011 - 12:57.

Hello I am using the Atmega 16 for receiver and transmitter is connected to the Atmega 8 and trying to light the LED like you to check whether my transmitters and receivers are working correct or not. But what I have got is that the transmitter is transmitting the data continuously as the LED connected to the transmitter pin is blinking continuously and the receiver pin LED is just on throughout, and I am not getting the output I mean the LED connected to my C0 pin is not blinking as hoped, so can you give me any idea why is this so. Thank you :cry:

- [reply](#)

Add new comment