Implementing RF Modules

A while ago I bought an RF link pair(which consists of one RF transmitter and one RF receiver) for Sparkfun for implementation in one of my client's products. I wanted to try out RF because it was cheaper than bluetooth. And by cheaper I mean A LOT cheaper. A bluetooth module would have cost me around $40 and a single RF receiver costs around $4. I had a lot of problems when I first started using the RF modules but after weeks of struggling with the RF modules , I managed to perfect it. This tutorial is meant to teach you how to use the RF link pair and help you out if you run into trouble with them.

This tutorial helped me out a bit - http://winavr.scienceprog.com/example-avr-projects/running-tx433-and-rx433-rf-modules-with-avr-microcontrollers.html . I will try to go into more detail than that tutorial and make it easier to understand.

First off, this tutorial is for the ATmega168 microcontroller, produced by Atmel. For other AVR microcontrollers you might need to do a little modifications to the code. Also this code will use the C language , using the GCC compiler. Enough of introduction, lets begin the tutorial:

Radio Frequency (RF) communication has a ton of applications. It can be used in robots, home automation, special effects, or in any application that needs the wireless transfer of data. The data transfer speed varies based on the receiver and transmitter , but the ones that I will be using throughout the rest of the tutorial are rated for 2400 baud, or 300 bytes per second. In the real world, with all the electrical interference, I get around 240 bytes per second. However, since half of the 240 bytes per second is used for encoding, I really can only do 120 bytes of data a second. Sparkfun says the range of the RF link is 500 ft. In reality the range is closer to 250 feet .We will deal with encoding and data transmission later on.

The RF link consists of one RF transmitter and one RF receiver. Make sure the transmitter and receiver are both for the same frequency. The transmitter can be bought here : http://www.sparkfun.com/commerce/product_info.php?products_id=8945 . The receiver can be bought here: http://www.sparkfun.com/commerce/product_info.php?products_id=8947

Transmitter looks like this: 

The receiver has more pins and looks like this: 

Once you have obtained the transmitter and receiver its time to understand what is the function of each pin. Your RF transmitter and receiver will look like one of the following pictures :

or like 

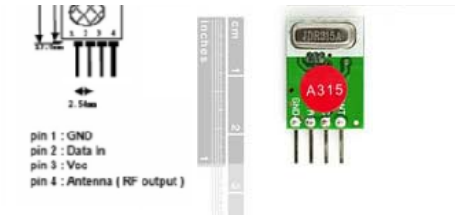
Both of the above pairs of RF links have the exact same pins and the exact same stuff. That means that they are virtually identical to each other in both appearance and function. You need one of the above pairs for this tutorial.

I will summarize the datasheets of the receiver and transmitter and write the most important information that you would need to know about the RF transmitter and receiver. Also you should note that a 23cm(around 9 inches) Antenna is recommended on both of them - just connect any piece of 23 cm long to the Antenna pin.

Transmitter:

Datasheet - http://www.sparkfun.com/datasheets/Wireless/General/MO-SAWR.pdf

The transmitter has 4 pins and a diagram is included to show pin function.

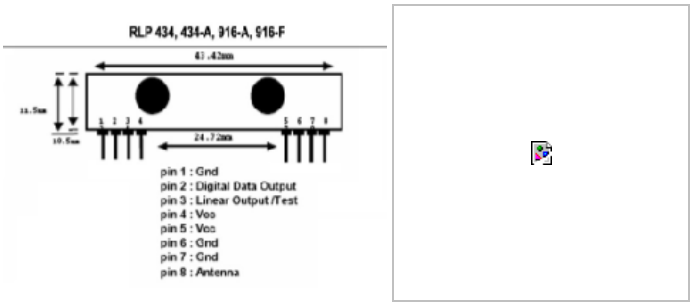


The transmitter can be powered by any voltage(Vcc) between 1.5V and 12V. The higher the voltage , the stronger the RF signal becomes. Just make sure not to exceed 12V because if you do the transmitter will break, which is bad. The data that the microcontroller will be sending will go into the pin labeled "Data In" .We also learn in the datasheet that the transmitter needs 20 milliseconds to start up.

Receiver:

Datasheet- http://www.sparkfun.com/datasheets/Wireless/General/MO-RX3400.pdf

The receiver has 8 pins and the datasheet also provides us with a diagram.



The receiver needs between 4.5V and 5.5V to power it. This means we will be feeding the Vcc pin of the receiver (the pin that you use to power up the receiver) a regulated 5V supply. The receiver will output any data it receives through Pin 2 , which is labeled "Digital Data Output". Pin 3, labeled "Linear Output/Test" is for testing out the receiver, and we will not be using it at all. It takes the receiver around 30 milliseconds to start up.

Introduction to the Data Transmission

As you know by know data is sent into the RF transmitter and then the RF receiver receives that exact data that was sent and sends the data out. In other words, the RF transmitter has DataIn and the RF receiver has DataOut. The data protocol that we will be using when sending and receiving is called UART. This tutorial has some great information on UART - http://www.societyofrobots.com/microcontroller_uart.shtml

One of the few things we need to set with UART is the baud rate. This is basically the data transmission rate. If we look at the RF transmitter and receiver specs we see that the maximum baud rate is 4800. We're going to play on the safe side and use one baud rate lower than 4800 , which is 2400. Using a baud rate of 2400 means that we get a maximum of 2400 bits per second which is equal to 300 bytes per second. Like I said before, you really end up getting only 240 bytes per second.

The real transmission rate of 240 bytes per second is further bogged down to 120 bytes of data a second because of encoding. Since there is a ton of electrical noise and radio frequency noise in most urban environment , you need something called encoding to filter out the electrical noise (the bad stuff) from teh data thats being sent (the good stuff). Encoding is basically "password protecting" your data. Think of it as putting a password on your data as you send it out through the transmitter. Then when the receiver receives any data at all (this would include electrical noise, radio frequency noise, and the correct data) it checks to see if the data has the correct "password" sent with it. If it does have the correct password then the data is accepted.

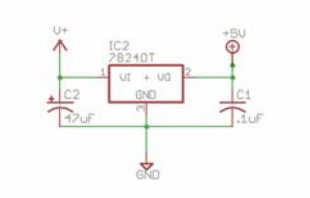
We will be using the ATmega168 as the microcontroller in this tutorial. The microcontroller is responsible for sending out the UART data to the transmitter and another microcontroller is responsible for receiving the UART data from the receiver and filtering out the bad data from the good data (using the "password" method of encoding). I use regular C with the GCC compiler on the ATmega168. I also like to use the AVRlib libraries , info here: http://www.mil.ufl.edu/~chrisarnold/components/microcontrollerBoard/AVR/avrlib/docs/html/ .

We will get into the technical stuff like the the program code later on in the tutorial.

Setting up the 5V Logic Power Supply

Since microcontrollers need a regulated 5V power source we will make one ourselves. It is extremely easy . All you need is a 5V Voltage Regulator chip (common ones are the 7805) and if you want you can add two capacitors to smooth out the current - its recommended but not necessary.

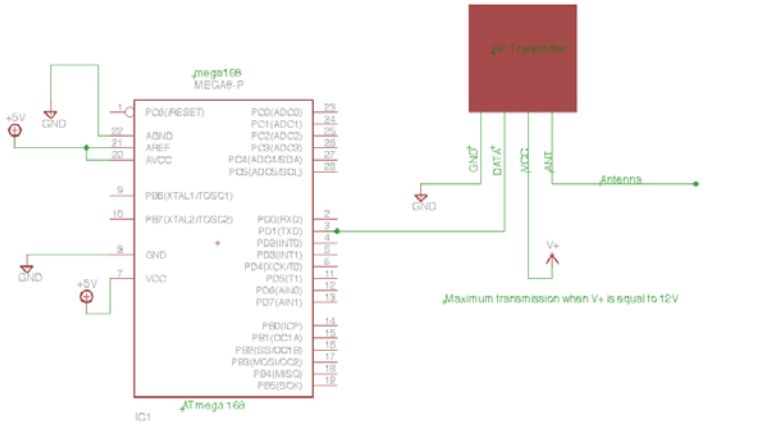
Heres a schematic and a picture



Creating the Transmitter Circuit

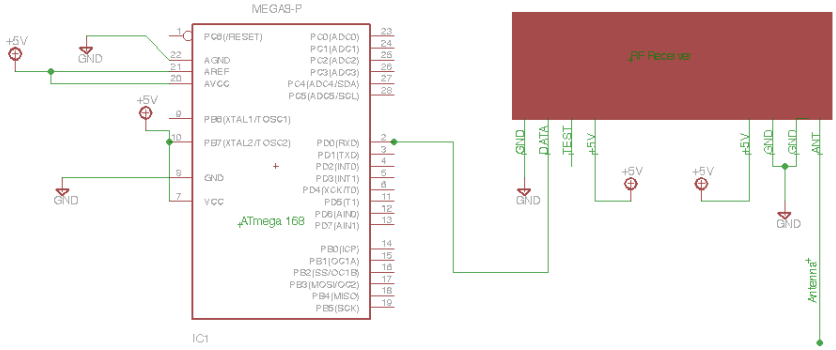
Creating the transmitter circuit is incredibly simple. All it involves is connecting the transmitter DataIn pin to the microcontroller DataOut pin (aka the TXD pin). There is one tricky part though . The Vcc of the transmitter should be connected to V+ , which is the original unregulated voltage. Remember when we were talking about how transmitter strength and range is based on how much voltage we give it? Well here is the pin for it. I personally give V+ the maximum voltage of 12V because I need as much range as possible.

Here is a schematic of the transmitter circuit :



Creating the Receiver Circuit

The Receiver circuit is also fairly simple . Just follow the schematic below. DataOut of the RF receiver goes to DataIn of the microcontroller (labeled pin RXD).



Download the Software Code here: RF_Modules_Source.zip

Notice in the main file "Photovore_v1.c" that there is the following code:

```
// Put a // before the subroutine that you DO NOT WANT TO RUN
// Delete the // before the subroutine that you DO WANT TO RUN

receive(); //Go to the subroutine responsible for receiving, it is located in "rf_rx.c"
// transmit(); //Go to the subroutine responsible for transmitting, it is located in "rf_rx.c"
```

This part is responsible for choosing which program is run , the receive or the transmit.

Transmitter Code:

In Photovore_v1.c it should have this code

```
// Put a // before the subroutine that you DO NOT WANT TO RUN
// Delete the // before the subroutine that you DO WANT TO RUN

//receive(); //Go to the subroutine responsible for receiving, it is located in "rf_rx.c"
transmit(); //Go to the subroutine responsible for transmitting, it is located in "rf_rx.c"
```

Initiliazing and Setting the Baud Rate

```
uartInit(); // initialize the UART (serial port)
uartSetBaudRate(2400);// set the baud rate of the UART for our debug/reporting output
```

This is where we initialize the UART and set the baud rate to 2400 bps.

SendByte Routine

```
void SendByte(char c) {
// Syntax is SendByte(#); where # is any number
while (!(UCSR0A & (1<<UDRE0))); // Wait until you are able to send a byte
UDR0 = c; // pass the value of the c variable into the UDR ,

// which in turn sends the byte out the UART
}
```

This is a pretty complicated piece of code. Basically what happens is that when you use the SendByte routine you are putting a byte into the UDR register which is in charge of sending out bytes out the UART.

Sending Packets

```
SendByte(start); // Send start byte
SendByte(data); // Send data byte
SendByte(addr); // Send address byte
SendByte(chksum); // Send checksum byte
delay_ms(2); // delay for 2 milliseconds
```

My packet consists of four bytes being sent with a 2 millisecond delay in between packets. For some reason if there is a delay less than 2 milliseconds the RF doesn't work properly , so I just added in a 2ms delay.

Receiver Code:

In Photovore_v1.c it should have this

```
// Put a // before the subroutine that you DO NOT WANT TO RUN
// Delete the // before the subroutine that you DO WANT TO RUN

receive(); //Go to the subroutine responsible for receiving, it is located in "rf_rx.c"
//transmit(); //Go to the subroutine responsible for transmitting, it is located in "rf_rx.c"
```

Notice how I put comment marks on receive() and removed the comment marks on transmit.

GetByte Routine:

```
int GetByte(void)
{
//Syntax is var= GetByte(); where var is any variable capable of storing a byte
while((UCSR0A&(1<<RXC0)) == 0); // wait until a byte is received
return UDR0; // once a byte is received send back to the program the byte
}
```

This subroutine is responsible for retrieving the byte from the UDR register which receives bytes from the UART.

Receiving Packets:

```
//Sample program for receiving packet with contents - 0x55, 0x8E ,0x27,0xB3

start = GetByte(); // Store received UART byte into the variable "start"
if (start == 0x55) // if data is equal to 0x55 , Start Byte
{
UDR =0; //clear UDR
data = GetByte(); // Store received UART byte into the variable "data"

UDR =0; //clear UDR
```

```
addr = GetByte(); // Store received UART byte into the variable "addr"
UDR =0; //clear UDR
chksum = GetByte(); // Store received UART byte into the variable "chksum"
if (chksum == (data+addr)) //if chksum is equal to data+addr then
{
if (data == 0x27) // if the data received is equal to 0x27
{
LED_on(); //turn on LED
} //endif
}//endif
}//endif
```

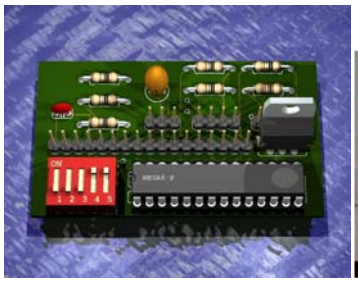
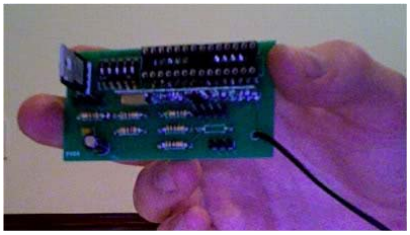
Modifying the UART.c AVRlib file

I am a big fan of AVRlib and I wanted to use some of AVRlib's uart library but not all of the subroutines. In this tutorial the only AVRlib UART routines that we will use is the uartInit and the uartSetBaudRate . The only modification that I had to do to the UART file was to disable all the UART interrupts so that my custom routines (GetByte and SendByte) would work properly.

The modified uart.h and uart.c files can be downloaded from here : Modified_UART.zip

Taking it Further:

Wireless RF can be added to almost any application. For one project, I needed to communicate from a laptop to multiple devices wirelessly. Each device had a receiver and a microcontroller. The address of each device was chosen externally by a DIP switch. Here are some pictures of the receiver:

I also created a RF dongle for the RF transmitter.

Here is a 3D model of a preliminary version of it:

You plug in the USB device into a USB port. Then you plug in the RF transmitter into a socket. Open up a Terminal window and whatever you send out the USB device's virtual serial port is sent out the RF transmitter.

The RF dongle that I designed will be on sale in about a month. Look for it on the narobo.com Products page.

,Eric

narobo.com