VirtualWire

Mike McCauley

Documentation for the VirtualWire communications library for Arduino.

1.0 Introduction

Arduino is a low cost microcontroller with Open Source hardware, see http://www.arduino.cc. VirtualWire is a communications library for Arduino that allows multiple Arduino's to communicate using low-cost RF transmitters and receivers.

2.0 Overview

VirtualWire is an Arduino library that provides features to send short messages, without addressing, retransmit or acknowledgment, a bit like UDP over wireless, using ASK (amplitude shift keying). Supports a number of inexpensive radio transmitters and receivers. All that is required is transmit data, receive data and (for transmitters, optionally) a PTT transmitter enable.

It is intended to be compatible with the RF Monolithics (www.rfm.com) Virtual Wire protocol, but this has not been tested.

Does not use the Arduino UART. Messages are sent with a training preamble, message length and checksum. Messages are sent with 4-to-6 bit encoding for good DC balance, and a CRC checksum for message integrity.

Why not just use the Arduino UART connected directly to the transmitter/receiver? As discussed in the RFM documentation, ASK receivers require a burst of training pulses to synchronize the transmitter and receiver, and also requires good balance between 0s and 1s in the message stream in order to maintain the DC balance of the message. UARTs do not provide these. They work a bit with ASK wireless, but not as well as this code.

2.1 Supported hardware.

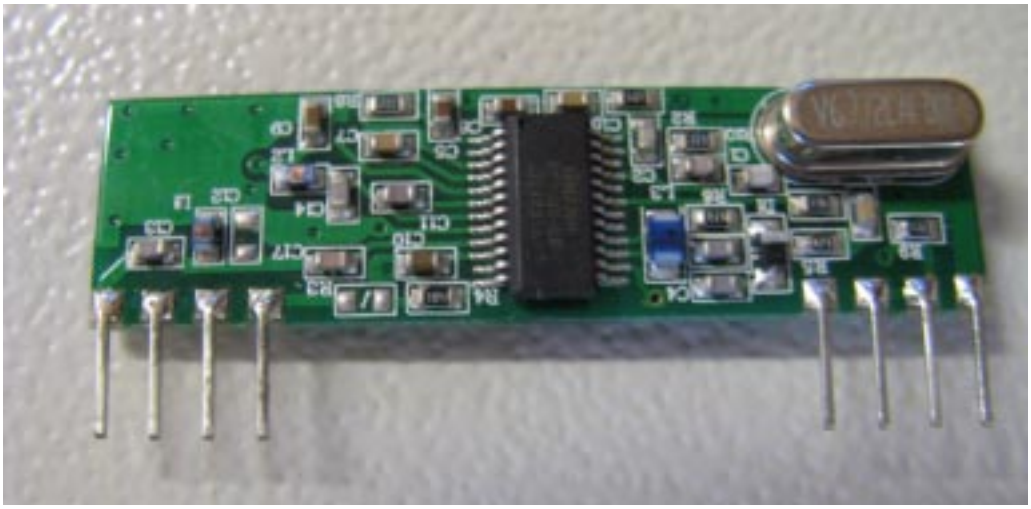
A range of communications hardware is supported. The ones listed blow are available in common retail outlets in Australian and other countries for under $10 per unit. Many other modules may also work with this software.

Runs on ATmega8/168 (Arduino Diecimila etc) and ATmega328 and possibly others.

2.2 Receivers

- RX-B1 (433.92MHz) (also known as ST-RX04-ASK)

FIGURE 1. RX-B1

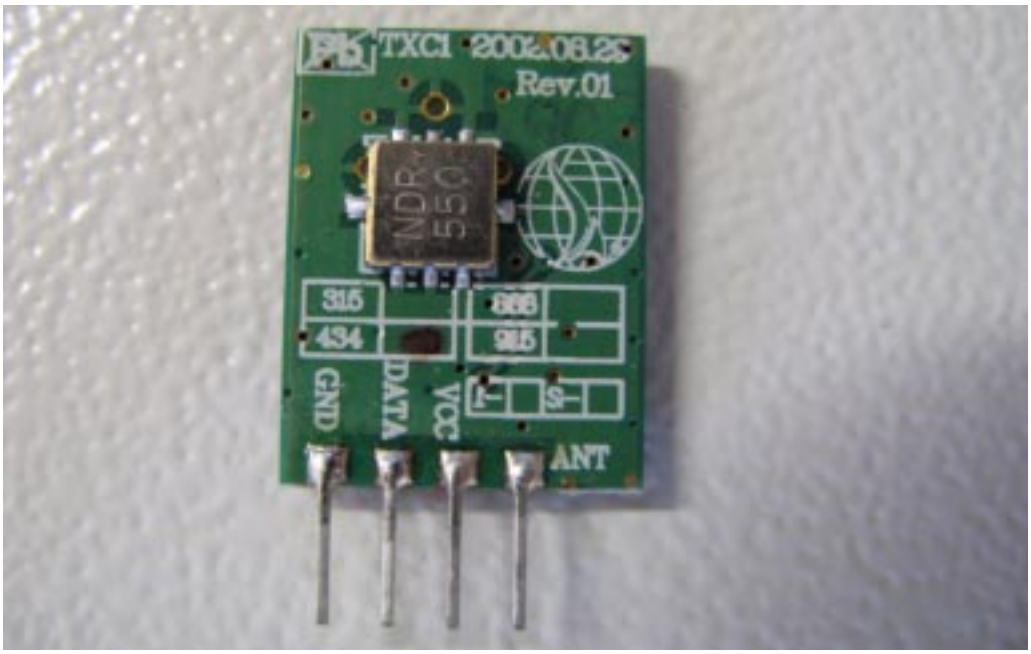


Details at http://www.summitek.com.tw/ST_SPEC/ST-RX04-ASK.pdf

2.3 Transmitters:

- TX-C1 (433.92MHz)

FIGURE 2. TX-C1



Details at http://www.tato.ind.br/files/TX-C1.pdf

2.4 Transceivers:

- DR3100 (433.92MHz)

FIGURE 3. DR3100

Details at http://www.rfmonolithics.com/products/data/dr3100.pdf

3.0 Downloading and installation

The latest version of this document is available from

```
http://www.airspayce.com/mikem/arduino/VirtualWire.pdf
```

For detailed API documentation and download instructions, see

```
http://www.airspayce.com/mikem/arduino/VirtualWire
```

You can also find online help and disussion at

```
http://groups.google.com/group/virtualwire
```

4.0 Implementation Details

Messages of up to VW_MAX_PAYLOAD (27) bytes can be sent

Each message is transmitted as:

- 36 bit training preamble consisting of 0-1 bit pairs
- 12 bit start symbol 0xb38

- 1 byte of message length byte count (4 to 30), count includes byte count and FCS bytes
- n message bytes, maximum n is VW_MAX_PAYLOAD (27)
- 2 bytes FCS, sent low byte-hi byte

Everything after the start symbol is encoded 4 to 6 bits, Therefore a byte in the message is encoded as 2x6 bit symbols, sent hi nybble, low nybble. Each symbol is sent LSBit first.

The Arduino Diecimila clock rate is 16MHz => 62.5ns/cycle.

For an RF bit rate of 2000 bps, need 500microsec bit period.

The ramp requires 8 samples per bit period, so need 62.5microsec per sample => interrupt tick is 62.5microsec.

The maximum packet length consists of

(6 + 2 + VW_MAX_MESSAGE_LEN*2) * 6 = 408 bits = 0.204 secs (at 2000 bps).

where VW_MAX_MESSAGE_LEN is VW_MAX_PAYLOAD + 3 (= 30).

The code consists of an ISR interrupt handler. Most of the work is done in the interrupt handler for both transmit and receive, but some is done from the user level. Expensive functions like CRC computations are always done in the user level.

Caution: VirtualWire takes over Arduino Timer1, and this will affect the PWM capabilities of the digital pins 9 and 10.

5.0 Performance

Unit tests show that the receiver PLL can stand up to 1 sample in 11 being inverted by noise without ill effect.

Testing with TX-C1, RX-B1, 5 byte message, 17cm antenna, no ground plane, 1m above ground, free space

At 10000 bps the transmitter does not operate correctly (ISR running too frequently at 80000/sec?)

At 9000 bps, asymmetries in the receiver prevent reliable communications at any distance

At 7000bps, Range about 90m

At 5000bps, Range about 100m

At 2000bps, Range over 150m

At 1000bps, Range over 150m

As suggested by RFM documentation, near the limits of range, reception is strongly influenced by the presence of a human body in the signal line, and by module orientation.

Throughout the range there are nulls and strong points due to multipath reflection. So... your mileage may vary.

Similar performance figures were found for DR3100. 9000bps worked.

Arduino and TX-C1 transmitter draws 27mA at 9V.

Arduino and RX-B1 receiver draws 31mA at 9V.

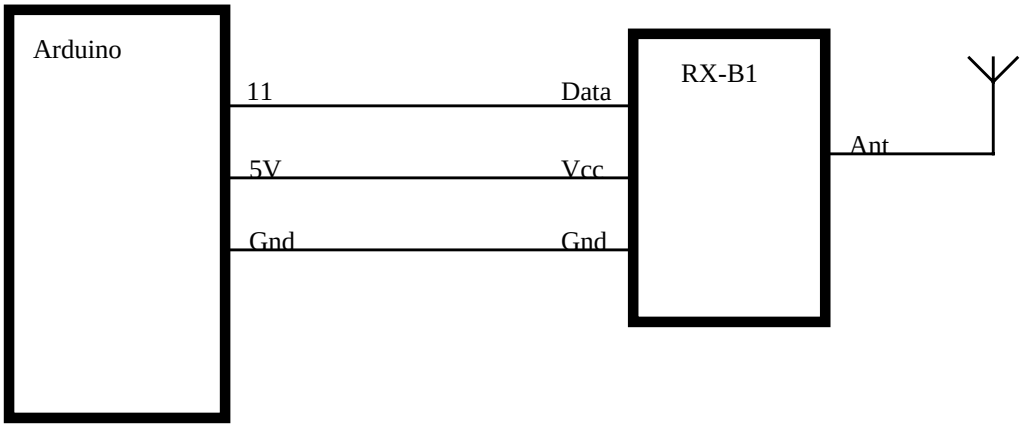
Arduino and DR3100 receiver draws 28mA at 9V.

6.0 Connections

Note that the IO pins can be changed from their defaults (as shown here) to any suitable IO pins using the vw_set_*_pin() calls.

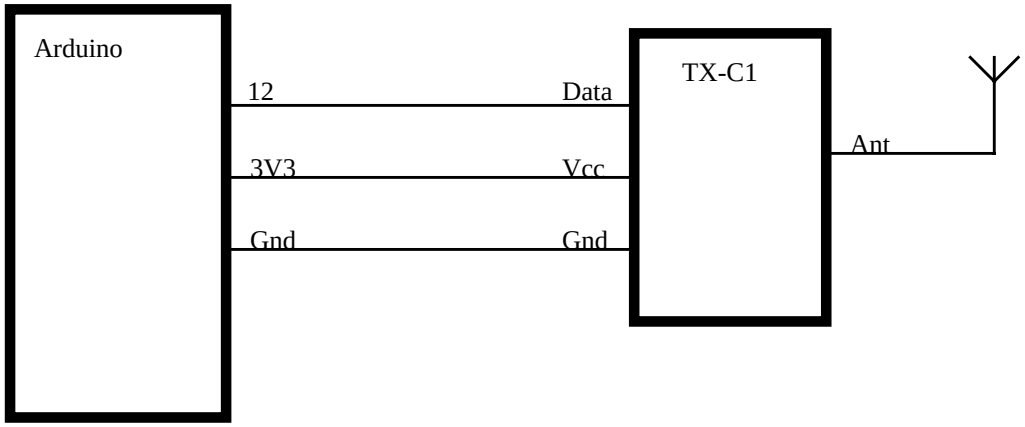
6.1 RX-B1 receiver

FIGURE 4. Wiring for RX-B1 receiver



6.2 TX-C1 transmitter

FIGURE 5. Wiring for TX-C1 transmitter



6.3 DR3100 transceiver

FIGURE 6. Wiring for DR3100

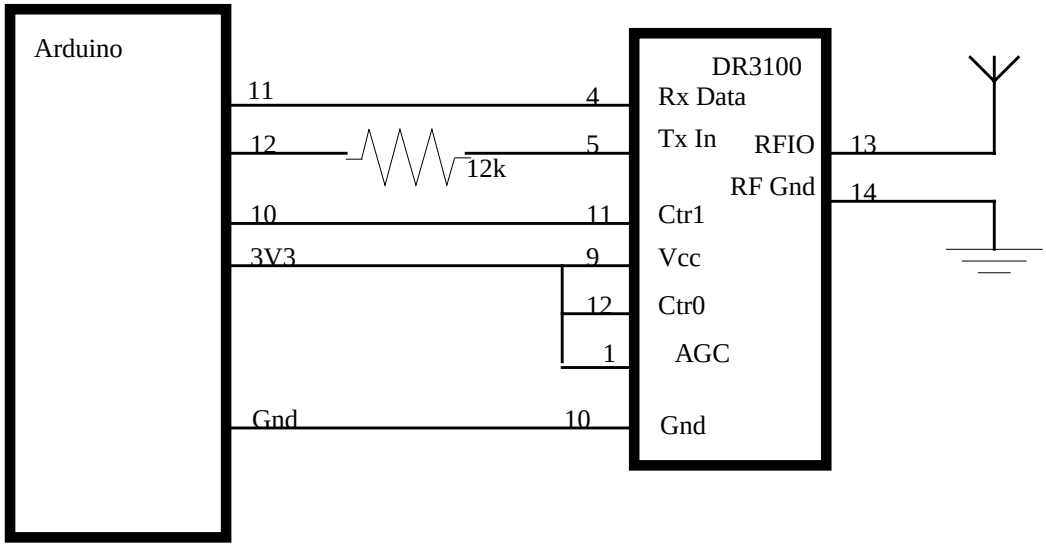
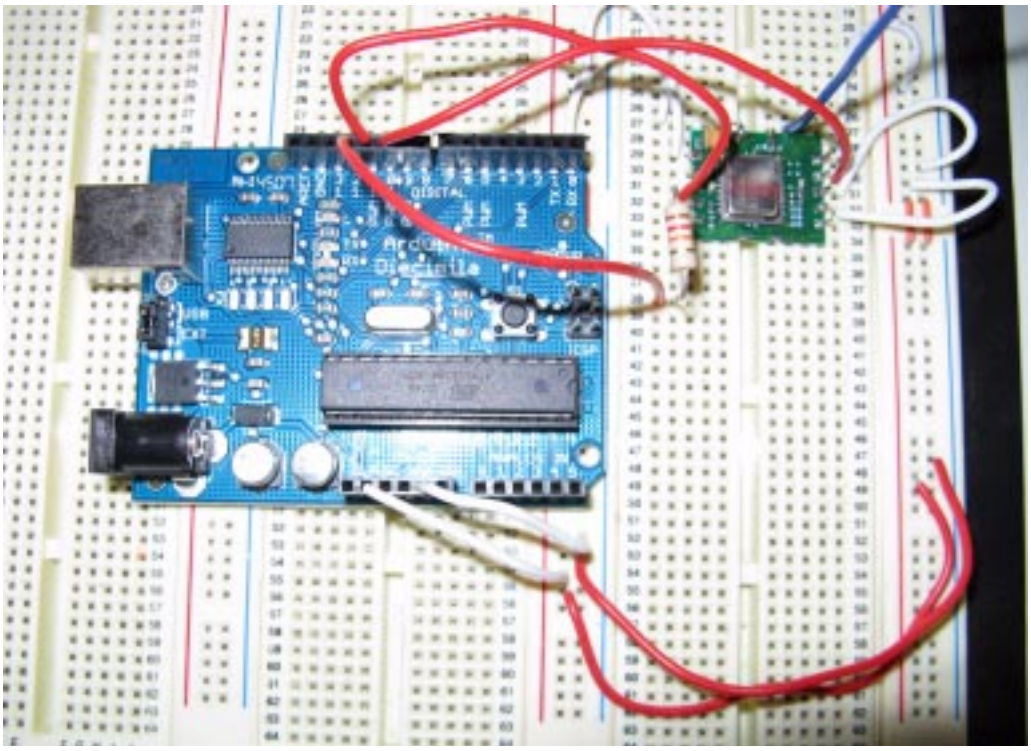


FIGURE 7. DR3100 connections on breadboard



Connections shown for no AGC, 1mW power out, 2400bps. Note the 12k resistor in series with Tx In to control the power output If you want to use higher data rates, need a resistor from pin 8 of the DR3100 to ground (see RFM documentation for more details).

If you want to use AGC, need a capacitor from pin 1 of the DR3100 to ground (see RFM documentation for more details).

The DR3100 module is supplied without any connection pins, only surface mount style pads. You will need to solder pins onto the module if you wish to use it in a breadboard.

7.0 Copyright and License

This software is Copyright (C) Mike McCauley. Use is subject to license conditions. The main licensing options available are GPL V2 or Commercial:

7.1 Open Source Licensing GPL V2

This is the appropriate option if you want to share the source code of your application with everyone you distribute it to, and you also want to give them the right to share who uses it. If you wish to use this software under Open Source Licensing, you must contrib-

ute all your source code to the open source community in accordance with the GPL Version 2 when your application is distributed. See http://www.gnu.org/copyleft/gpl.html

7.2 Commercial Licensing

This is the appropriate option if you are creating proprietary applications and you are not prepared to distribute and share the source code of your application. Contact info@airspayce.com for details.