

✓

Join GitHub today

GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

<>

Arduino library for data transmission using common 315Mhz and 433Mhz transmitters using Manchester encoding. Good for using with microcontrollers with internal clock where clock accuracy cannot be guaranteed. Works on all common AVR's and speeds

51 commits

1 branch

0 releases

7 contributors

Branch: master ▾

New pull request


Find file

Clone or download ▾

This branch is 5 commits ahead, 33 commits behind mchr3k:master.

Pull request

Compare



cano64 Merge pull request #2 from borodaxel/master ...

Latest commit 454300e on 21 Mar 2016

<div><div></div>examples</div>	Fixed receiveArray function and its declaration. Now it points to the...	2 years ago
<div><div></div>ManchesterRF.cpp</div>	Fixed receiveArray function and its declaration. Now it points to the...	2 years ago
<div><div></div>ManchesterRF.h</div>	Fixed receiveArray function and its declaration. Now it points to the...	2 years ago
<div><div></div>README.markdown</div>	2.0 beta	4 years ago
<div><div></div>keywords.txt</div>	2.0 beta	4 years ago

README.markdown

This is a [Manchester encoding](#) RF library which works on Arduino and ATTiny.

Library supports:

various microcontrollers

- ATmega1284
- ATmega328
- ATmega8
- ATMega32U4
- ATtiny84 (44, 24)
- ATtiny85 (45, 85)

Various microcontroller's speed both RX and TX All microcontroller speeds between 1Mhz and 20Mhz are supported. The library has been tested with the following common speeds:

- 1Mhz
- 8Mhz
- 16Mhz

Various transmission speeds, the maximum transmission speed is mostly dependent on your microcontroller speed

- 300baud
- 600baud
- 1200baud
- 2400baud

- 4800baud
- 9600baud
- 19200baud

With this library you can sucessfully transmit data between various microcontrollers runnning at various speeds even if their clock speed drifts up to 100%. It's specifically designed to work with innacurate internal oscilator.

Speed

Transmition speed is determined by the speed of the microcontroller. Following speeds can be achieved using direct port manipulation. Receiver: freq -> max receive speed 1Mhz -> 1200baud 8Mhz -> 9600baud 16Mhz -> 19200baud

Preserving PWM

Although this library is using timers, PWM functionality related to that timer will be preserved, PWM frequency will increase though. Timer prescaler will be set to 1 (no prescaler) and therefore PWM frequency will be $F_{CPU}/256$.

Direct port manipulation

Standard Arduino digitalRead() and digitalWrite() functions are extremely slow. For faster comminication it is necessary to use direct port manipulation.

Notes about transmitters

<http://blog.solidremote.com/post/rf-module-external-antenna-design.aspx>

The library has been tested with common 315Mhz and 433Mhz transmitters using ASK OOK. Tips to improve your transmit distance: Attaching an antenna will greatly improve transmit distance (from few cm to few m) for 315Mhz use 23.81 cm straight wire, for 433Mhz use 17.28cm straight wire as antenna. If possible keep the wire straight, away from ground wire, and away from the rest of the circuit and battery pack. Transmitter can use anything from 3.3V to 12V. Increasing voltage will increase transmit distance. If you are using voltage regulator, attach transmitter directly to the battery. Receiver needs 5V, it doesn't work on 3.3V

Speed: I was able to achieve 19200 bauds between two 16Mhz Arduinos, or 2400 bauds between two 1Mhz ATTiny85, you can try different speeds to see which works the best for you.

Full duplex: for bidirectional communication use both 315Mhz and 433Mhz transmitters. This way they can transmit at the same time without interfering with each other. If you have just one type, wait for receiver to finish receiving before transmitting.

Credits

- Library originally from [carl47](#) on [this thread](#)
- Contributions from [mchr3k](#), [Mike Cano](#)