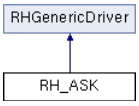
# RH_ASK Class Reference

Driver to send and receive unaddressed, unreliable datagrams via inexpensive ASK (Amplitude Shift Keying) or OOK (On Off Keying) RF transceivers. More...

`#include <RH_ASK.h>`

Inheritance diagram for RH_ASK:




## Public Member Functions

| | |
|---|---|
| | **RH_ASK** (uint16_t **speed**=2000, uint8_t rxPin=11, uint8_t txPin=12, uint8_t pttPin=10, bool pttInverted=false) |
| virtual bool | **init** () |
| virtual bool | **available** () |
| virtual RH_INTERRUPT_ATTR bool | **recv** (uint8_t *buf, uint8_t *len) |
| virtual bool | **send** (const uint8_t *data, uint8_t len) |
| virtual uint8_t | **maxMessageLength** () |
| RH_INTERRUPT_ATTR void | **setModeIdle** () |
| RH_INTERRUPT_ATTR void | **setModeRx** () |
| void | **setModeTx** () |
| RH_INTERRUPT_ATTR void | **handleTimerInterrupt** ()<br>dont call this it used by the interrupt handler |
| uint16_t | **speed** () |

**Public Member Functions inherited from RHGenericDriver**

## Protected Member Functions

| | |
|---|---|
| uint8_t | **timerCalc** (uint16_t **speed**, uint16_t max_ticks, uint16_t *nticks)<br>Helper function for calculating timer ticks. |
| void | **timerSetup** ()<br>Set up the timer and its interrutps so the interrupt handler is called at the right frequency. |
| RH_INTERRUPT_ATTR bool | **readRx** ()<br>Read the rxPin in a platform dependent way, taking into account whether it is inverted or not. |
| void | **writeTx** (bool value)<br>Write the txPin in a platform dependent way. |
| void | **writePtt** (bool value)<br>Write the txPin in a platform dependent way, taking into account whether it is inverted or not. |
| RH_INTERRUPT_ATTR uint8_t | **symbol_6to4** (uint8_t symbol)<br>Translates a 6 bit symbol to its 4 bit plaintext equivalent. |
| void | **receiveTimer** ()<br>The receiver handler function, called a 8 times the bit rate. |
| void | **transmitTimer** ()<br>The transmitter handler function, called a 8 times the bit rate. |
| void | **validateRxBuf** () |

## Protected Attributes

| | |
|---|---|
| uint16_t | **_speed**<br>Configure bit rate in bits per second. |
| uint8_t | **_rxPin**<br>The configure receiver pin. |
| uint8_t | **_txPin**<br>The configure transmitter pin. |
| uint8_t | **_pttPin**<br>The configured transmitter enable pin. |
| bool | **_rxInverted**<br>True of the sense of the rxPin is to be inverted. |
| bool | **_pttInverted** |

| | |
|---|---|
| | True of the sense of the pttPin is to be inverted. |
| volatile bool | **_rxBufFull**<br>Buf is filled but not validated. |
| volatile bool | **_rxBufValid**<br>Buf is full and valid. |
| volatile bool | **_rxLastSample**<br>Last digital input from the rx data pin. |
| volatile uint8_t | **_rxIntegrator** |
| volatile uint8_t | **_rxPllRamp** |
| volatile uint8_t | **_rxActive** |
| volatile uint16_t | **_rxBits**<br>Last 12 bits received, so we can look for the start symbol. |
| volatile uint8_t | **_rxBitCount**<br>How many bits of message we have received. Ranges from 0 to 12. |
| uint8_t | **_rxBuf** [RH_ASK_MAX_PAYLOAD_LEN]<br>The incoming message buffer. |
| volatile uint8_t | **_rxCount**<br>The incoming message expected length. |
| volatile uint8_t | **_rxBufLen**<br>The incoming message buffer length received so far. |
| uint8_t | **_txIndex**<br>Index of the next symbol to send. Ranges from 0 to vw_tx_len. |
| uint8_t | **_txBit**<br>Bit number of next bit to send. |
| uint8_t | **_txSample**<br>Sample number for the transmitter. Runs 0 to 7 during one bit interval. |
| uint8_t | **_txBuf** [(RH_ASK_MAX_PAYLOAD_LEN *2)+RH_ASK_PREAMBLE_LEN]<br>The transmitter buffer in *symbols* not data octets. |
| uint8_t | **_txBufLen**<br>Number of symbols in _txBuf to be sent;. |

▸ **Protected Attributes inherited from RHGenericDriver**

## Additional Inherited Members

▸ **Public Types inherited from RHGenericDriver**

Defines different operating modes for the transport hardware. More...

▸ **Static Public Member Functions inherited from RHGenericDriver**

## Detailed Description

Driver to send and receive unaddressed, unreliable datagrams via inexpensive ASK (Amplitude Shift Keying) or OOK (On Off Keying) RF transceivers.

The message format and software technology is based on our earlier VirtualWire library (http://www.airspayce.com/mikem/arduino/VirtualWire), with which it is compatible. See http://www.airspayce.com/mikem/arduino/VirtualWire.pdf for more details. VirtualWire is now obsolete and unsupported and is replaced by this library.

**RH_ASK** is a Driver for Arduino, Maple and others 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. Can also be used over various analog connections (not just a data radio), such as the audio channel of an A/V sender, or long TTL lines.

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.

But why not just use a 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.

**Theory of operation**

See ASH Transceiver Software Designer's Guide of 2002.08.07 http://wireless.murata.com/media/products/apnotes/tr_swg05.pdf?ref=rfm.com

http://web.engr.oregonstate.edu/~moon/research/files/cas2_mar_07_dpll.pdf while not directly relevant is also interesting.

**Implementation Details**

Messages of up to RH_ASK_MAX_PAYLOAD_LEN (67) 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 (uincluding 4 bytes of header), maximum n is RH_ASK_MAX_MESSAGE_LEN + 4 (64)
- 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 message may consist of any binary digits.

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 + RH_ASK_MAX_MESSAGE_LEN*2) * 6 = 768 bits = 0.384 secs (at 2000 bps). where RH_ASK_MAX_MESSAGE_LEN is RH_ASK_MAX_PAYLOAD_LEN - 7 (= 60). 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.

**Supported Hardware**

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

Runs on a wide range of Arduino processors using Arduino IDE 1.0 or later. Also runs on on Energia, with MSP430G2553 / G2452 and Arduino with ATMega328 (courtesy Yannick DEVOS - XV4Y), but untested by us. It also runs on Teensy 3.0 (courtesy of Paul Stoffregen), but untested by us. Also compiles and runs on ATtiny85 in Arduino environment, courtesy r4z0r7o3. Also compiles on maple-ide-v0.0.12, and runs on Maple, flymaple 1.1 etc. Runs on ATmega8/168 (Arduino Diecimila, Uno etc), ATmega328 and can run on almost any other AVR8 platform, without relying on the Arduino framework, by properly configuring the library editing the **RH_ASK.h** header file for describing the access to IO pins and for setting up the timer. Runs on ChipKIT Core supported processors such as Uno32 etc.

- Receivers
  - RX-B1 (433.92MHz) (also known as ST-RX04-ASK)
  - RFM83C from HopeRF http://www.hoperfusa.com/details.jsp?pid=126
  - SYN480R and other similar ASK receivers
- Transmitters:
  - TX-C1 (433.92MHz)
  - RFM85 from HopeRF http://www.hoperfusa.com/details.jsp?pid=127
  - SYN115, F115 and other similar ASK transmitters
- Transceivers
  - DR3100 (433.92MHz)

**Connecting to Arduino**

Most transmitters can be connected to Arduino like this:

```
Arduino                        Transmitter
 GND----------------------------GND
 D12----------------------------Data
 5V-----------------------------VCC
```

Most receivers can be connected to Arduino like this:

```
Arduino                        Receiver
 GND----------------------------GND
 D11----------------------------Data
 5V-----------------------------VCC
                                 SHUT (not connected)
                                 WAKEB (not connected)
                                 GND |
                                 ANT |- connect to your antenna syetem
```

**RH_ASK** works with ATTiny85, using Arduino 1.0.5 and tinycore from https://code.google.com/p/arduino-tiny/downloads/detail?name=arduino-tiny-0100-0018.zip Tested with the examples ask_transmitter and ask_receiver on ATTiny85. Caution: The RAM memory requirements on an ATTiny85 are *very* tight. Even the bare bones ask_transmitter sketch barely fits in eh RAM available on the ATTiny85. Its unlikely to work on smaller ATTinys such as the ATTiny45 etc. If you have wierd behaviour, consider reducing the size of RH_ASK_MAX_PAYLOAD_LEN to the minimum you can work with. Caution: the default internal clock speed on an ATTiny85 is 1MHz. You MUST set the internal clock speed to 8MHz. You can do this with Arduino IDE, tineycore and ArduinoISP by setting the board type to "ATtiny85@8MHz', setting theProgrammer to 'Arduino as ISP' and selecting Tools->Burn Bootloader. This does not actually burn a bootloader into the tiny, it just changes the fuses so the chip runs at 8MHz. If you run the chip at 1MHz, you will get RK_ASK speeds 1/8th of the expected.

Initialise **RH_ASK** for ATTiny85 like this:

```
// #include <SPI.h> // comment this out, not needed
RH_ASK driver(2000, 4, 3); // 200bps, TX on D3 (pin 2), RX on D4 (pin 3)
```

then: Connect D3 (pin 2) as the output to the transmitter Connect D4 (pin 3) as the input from the receiver.

For testing purposes you can connect 2 Arduino **RH_ASK** instances directly, by connecting pin 12 of one to 11 of the other and vice versa, like this for a duplex connection:

```
Arduino 1         wires         Arduino 1
 D11-----------------------------D12
 D12-----------------------------D11
 GND-----------------------------GND
```

You can also connect 2 **RH_ASK** instances over a suitable analog transmitter/receiver, such as the audio channel of an A/V transmitter/receiver. You may need buffers at each end of the connection to convert the 0-5V digital output to a suitable analog voltage.

Measured power output from RFM85 at 5V was 18dBm.

**ESP8266**

This module has been tested with the ESP8266 using an ESP-12 on a breakout board ESP-12E SMD Adaptor Board with Power Regulator from tronixlabs http://tronixlabs.com.au/wireless/esp8266/esp8266-esp-12e-smd-adaptor-board-with-power-regulator-australia/ compiled on Arduino 1.6.5 and the ESP8266 support 2.0 installed with Board Manager. CAUTION: do not use pin 11 for IO with this chip: it will cause the sketch to hang. Instead use constructor arguments to configure different pins, eg:

```
RH_ASK driver(2000, 2, 4, 5);
```

Which will initialise the driver at 2000 bps, recieve on GPIO2, transmit on GPIO4, PTT on GPIO5. Caution: on the tronixlabs breakout board, pins 4 and 5 may be labelled vice-versa.

**Timers**

The **RH_ASK** driver uses a timer-driven interrupt to generate 8 interrupts per bit period. **RH_ASK** takes over a timer on Arduino-like platforms. By default it takes over Timer 1. You can force it to use Timer 2 instead by enabling the define RH_ASK_ARDUINO_USE_TIMER2 near the top of RH_ASK.cpp On Arduino Zero it takes over timer TC3. On Arduino Due it takes over timer TC0. On ESP8266, takes over timer0 (which conflicts with ServoTimer0).

Caution: ATTiny85 has only 2 timers, one (timer 0) usually used for millis() and one (timer 1) for PWM analog outputs. The **RH_ASK** Driver library, when built for ATTiny85, takes over timer 0, which prevents use of millis() etc but does permit analog outputs. This will affect the accuracy of millis() and time measurement.

**STM32 F4 Discovery with Arduino and Arduino_STM32**

You can initialise the driver like this:

```
RH_ASK driver(2000, PA3, PA4);
```

and connect the serail to pins PA3 and PA4

## Constructor & Destructor Documentation

### ◆ RH_ASK()

```
RH_ASK::RH_ASK ( uint16_t  speed = 2000,
                 uint8_t   rxPin = 11,
                 uint8_t   txPin = 12,
                 uint8_t   pttPin = 10,
                 bool      pttInverted = false
               )
```

Constructor. At present only one instance of **RH_ASK** per sketch is supported.

**Parameters**

| | | |
|---|---|---|
| [in] | **speed** | The desired bit rate in bits per second |
| [in] | **rxPin** | The pin that is used to get data from the receiver |
| [in] | **txPin** | The pin that is used to send data to the transmitter |
| [in] | **pttPin** | The pin that is connected to the transmitter controller. It will be set HIGH to enable the transmitter (unless pttInverted is true). |
| [in] | **pttInverted** | true if you desire the pttin to be inverted so that LOW wil enable the transmitter. |

References **_txBuf**.

## Member Function Documentation

## ◆ available()

bool RH_ASK::available ( )

Tests whether a new message is available from the Driver. On most drivers, this will also put the Driver into RHModeRx mode until a r received bythe transport, when it wil be returned to RHModeIdle. This can be called multiple times in a timeout loop

**Returns**
true if a new, complete, error-free uncollected message is available to be retreived by **recv()**

Implements **RHGenericDriver**.

References **RHGenericDriver::_mode**, **_rxBufFull**, **_rxBufValid**, **RHGenericDriver::RHModeTx**, **setModeRx()**, and **validateRxB**

Referenced by **recv()**.

## ◆ init()

bool RH_ASK::init ( ) virtual

Initialise the Driver transport hardware and software. Make sure the Driver is properly configured before calling **init()**.

**Returns**
true if initialisation succeeded.

Reimplemented from **RHGenericDriver**.

References **_pttPin**, **_rxPin**, **_txPin**, **RHGenericDriver::init()**, **setModeIdle()**, and **timerSetup()**.

## ◆ maxMessageLength()

uint8_t RH_ASK::maxMessageLength ( ) virtual

Returns the maximum message length available in this Driver.

**Returns**
The maximum legal message length

Implements **RHGenericDriver**.

References **handleTimerInterrupt()**, and **speed()**.

## ◆ recv()

bool RH_INTERRUPT_ATTR RH_ASK::recv ( uint8_t * buf,
uint8_t * len
) virtual

Turns the receiver on if it not already on. If there is a valid message available, copy it to buf and return true else return false. If a message is copied, *len is set to the length (Caution, 0 length messages are permitted). You should be sure to call this function frequently enough to not miss any messages It is recommended that you call it in your main loop.

**Parameters**
`[in]` **buf** Location to copy the received message
`[in,out]` **len** Pointer to available space in buf. Set to the actual number of octets copied.

**Returns**
true if a valid message was copied to buf

Implements **RHGenericDriver**.

References **_rxBuf**, **_rxBufLen**, **_rxBufValid**, and **available()**.

## ◆ send()

bool RH_ASK::send ( const uint8_t * data,
uint8_t len
) virtual

Waits until any previous transmit packet is finished being transmitted with **waitPacketSent()**. Then loads a message into the transmitter and starts the transmitter. Note that a message length of 0 is NOT permitted.

**Parameters**

[in] **data** Array of data to be sent

[in] **len** Number of bytes of data to send (> 0)

**Returns**

true if the message length was valid and it was correctly queued for transmit

Implements **RHGenericDriver**.

References **_txBuf**, **_txBufLen**, **RHGenericDriver::_txHeaderFlags**, **RHGenericDriver::_txHeaderFrom**, **RHGenericDriver::_txHeaderId**, **RHGenericDriver::_txHeaderTo**, **setModeTx()**, **RHGenericDriver::waitCAD()**, and **RHGenericDriver::waitPacketSent()**.

## ◆ setModeIdle()

void RH_INTERRUPT_ATTR RH_ASK::setModeIdle ( )

If current mode is Rx or Tx changes it to Idle. If the transmitter or receiver is running, disables them.

References **RHGenericDriver::_mode**, **RHGenericDriver::RHModeIdle**, **writePtt()**, and **writeTx()**.

Referenced by **init()**, **receiveTimer()**, and **transmitTimer()**.

## ◆ setModeRx()

void RH_INTERRUPT_ATTR RH_ASK::setModeRx ( )

If current mode is Tx or Idle, changes it to Rx. Starts the receiver in the RF69.

References **RHGenericDriver::_mode**, **RHGenericDriver::RHModeRx**, **writePtt()**, and **writeTx()**.

Referenced by **available()**.

## ◆ setModeTx()

void RH_ASK::setModeTx ( )

If current mode is Rx or Idle, changes it to Rx. F Starts the transmitter in the RF69.

References **RHGenericDriver::_mode**, **_txBit**, **_txIndex**, **_txSample**, **RHGenericDriver::RHModeTx**, and **writePtt()**.

Referenced by **send()**.

## ◆ speed()

uint16_t RH_ASK::speed ( ) inline

Returns the current speed in bits per second

**Returns**

The current speed in bits per second

References **_speed**, **readRx()**, **receiveTimer()**, **symbol_6to4()**, **timerCalc()**, **timerSetup()**, **transmitTimer()**, **validateRxBuf()**, **writePtt()**, and **writeTx()**.

Referenced by **maxMessageLength()**.

### ◆ validateRxBuf()

void RH_ASK::validateRxBuf ( )

Check whether the latest received message is complete and uncorrupted We should always check the FCS at user level, not in slow

References **RHGenericDriver::_promiscuous**, **RHGenericDriver::_rxBad**, **_rxBuf**, **_rxBufLen**, **_rxBufValid**, **RHGenericD RHGenericDriver::_rxHeaderFlags**, **RHGenericDriver::_rxHeaderFrom**, **RHGenericDriver::_rxHeaderId**, **RHGenericD RHGenericDriver::_thisAddress**.

Referenced by **available()**, and **speed()**.

## Member Data Documentation

### ◆ _rxActive

volatile uint8_t RH_ASK::_rxActive protected

Flag indicates if we have seen the start symbol of a new message and are in the processes of reading and decoding it

Referenced by **receiveTimer()**.

### ◆ _rxIntegrator

volatile uint8_t RH_ASK::_rxIntegrator protected

This is the integrate and dump integral. If there are <5 0 samples in the PLL cycle the bit is declared a 0, else a 1

Referenced by **receiveTimer()**.

### ◆ _rxPllRamp

volatile uint8_t RH_ASK::_rxPllRamp protected

PLL ramp, varies between 0 and RH_ASK_RX_RAMP_LEN-1 (159) over RH_ASK_RX_SAMPLES_PER_BIT (8) samples per nominal bit time. When the PLL is synchronised, bit transitions happen at about the 0 mark.

Referenced by **receiveTimer()**.

The documentation for this class was generated from the following files:

- **RH_ASK.h**
- RH_ASK.cpp