



# Adding A Low Cost Wireless Packet Radio To

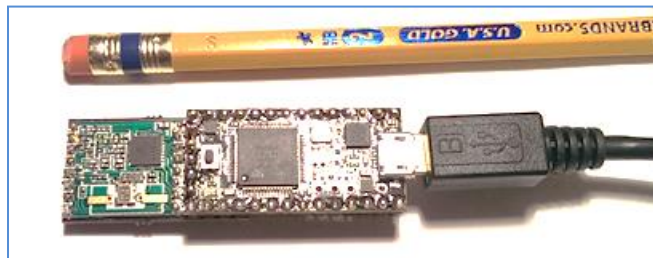
## **A Teensy 3 Microprocessor**

### **Plus: Software Diagnostic For Teensy or AVR mega32**

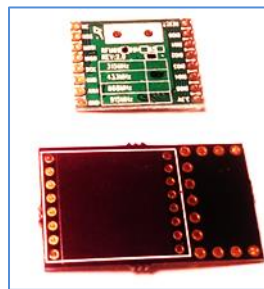
May 28, 2014, Steve Childress

#### **Goal**

Inexpensively enable a Teensy 3 microprocessor to send/receive packet data using a \$5 HopeRF RF69xx smart radio module. The wireless protocol software (freeware) for the Teensy 3 is RadioHead <http://www.airspayce.com/mikem/arduino/RadioHead/>. Teensy 3.1 is a low cost and very small ARM Cortex M4 based product from PJRC <http://www.pjrc.com/teensy/index.html> or <http://store.oshpark.com/products/teensy-3-1>. In addition to RadioHead, most any Arduino “sketch” program or library will run on the Teensy 3.1 via the Teensyduino library from PJRC (freeware).



Teensy 3 with RFM69 add-on radio (left-most). USB leads to PC for development.



Top: RFM69 Radio module. Available for 433, 868/915MHz bands. Up to 100mW. Cost \$5 or less.

Bottom: OSH Parks Adapter PC Board For RFM69 and Teensy 3

**Next: Hardware build; then Software Diagnostic.**

## Materials Needed, Per Device. Of course, 2+ devices needed.

1	Teensy 3.1 Arm Cortex M4 Microprocessor Board w/header pins. About \$17- \$20. Flash: 256KB; RAM: 64KB; 96MHz, 32 bit CPU.	<a href="http://www.pjrc.com/store/teensy31_pins.html">http://www.pjrc.com/store/teensy31_pins.html</a> or <a href="http://store.oshpark.com/products/teensy-3-1">http://store.oshpark.com/products/teensy-3-1</a> w/o pins-(you purchase and add standard header pins).
2	<a href="http://www.anarduino.com/?cid=4">http://www.anarduino.com/?cid=4</a> Browse for specific version of RFM69: by RF band (MHz). If powered by the Teensy 3's 3.3v pin, use the RFM69W. The RFM69HW (100mW) should be separately powered at 3.3V as it exceeds the 100mA capacity of the Teensy 3 pin.	Many sources, search w/google. Recommended source: Anarduino.com. About \$4 ea. Order spare(s). NOTE: choose module for either 433MHz or 868/915MHz. The latter tunes to the 902-928MHz ISM/Amateur radio band. The 433MHz module can tune to the 70cm Amateur band. Amateur bands require license; ISM do not. No. America and other regions have restrictions on transmitter on-time for ISM bands.
3	<a href="http://oshpark.com/shared_projects/Rlu mMBtN">http://oshpark.com/shared_projects/Rlu mMBtN</a>	Item: "rfm69_adaptor2". 3ea for \$4. Order spares. Designed and shared by Paul Stoffregen of PJRC, the Teensy 3's designer. Delivery time may be long.
4	Schmartboard #920-0018-01 Mouser #872-920-0018-01 Or ebay "2mm pin header"	2mm spacing pin header (breakaway to size). Used to mount RFM69 to adapter board. Cut excess length with small electronics style diagonal cutter pliers. <b>Alternative:</b> Use solid 26ga hookup wire and cut each.

## Tools needed

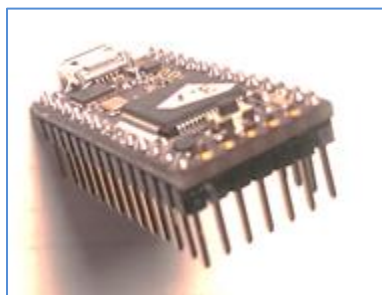
1. High quality soldering iron with small diameter tip (e.g., 2mm).
2. Small diameter solder, preferably flux core; use low-lead only if you are experienced with such
3. Pin headers, standard 0.1 inch/2.54mm. Plentiful supply
4. Common electronics needle-nose and wire clipper pliers

## Step 1

Test the Teensy 3.1 without the RFM69, using example programs.

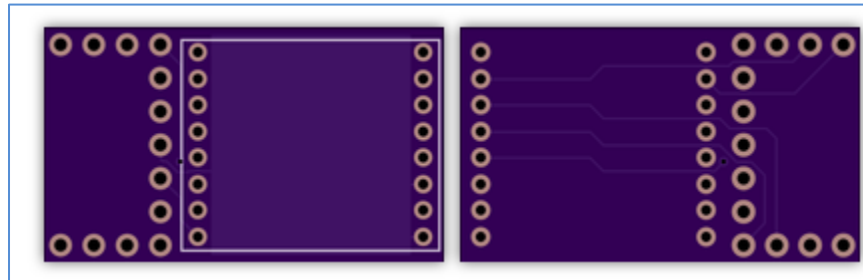
## Step 2

To the Teensy 3 w/pins, add and solder the pins on the small side of the teensy 3, opposite end from USB jack, as in the photo below



### Step 3

Study the RFM69 adapter board.

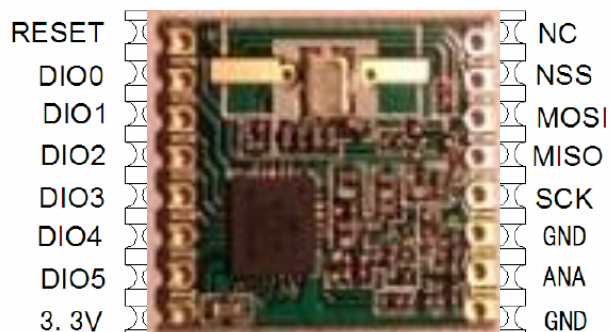


Above: RFM69 Adapter Board. Left is top view; right is bottom view. The RFM69 mounts on the top side. Through-holes on 2mm pitch (spacing) align with through-holes in the RFM69. Orient as shown in photo on the next page. Be sure to orient the radio correctly before soldering!!



Above: The pins on the Teensy 3 that mate with the adaptor board using either pins or pin/socket. The Teensy 3 board's edge connections for GND and 3.3V are used; A14, Program and VBat are not used by the adapter board. The table on the next page shows the use of the power and I/O pins.

When using 2mm pin headers to mount the RFM69 on the adapter board, the plastic insulator on the bank of pins must be removed to get proper spacing when attached to the Teensy. An alternative to 2mm pin headers is to use 26 ga. wire for each through-hole, or the holes needed as shown in the table on the next page.



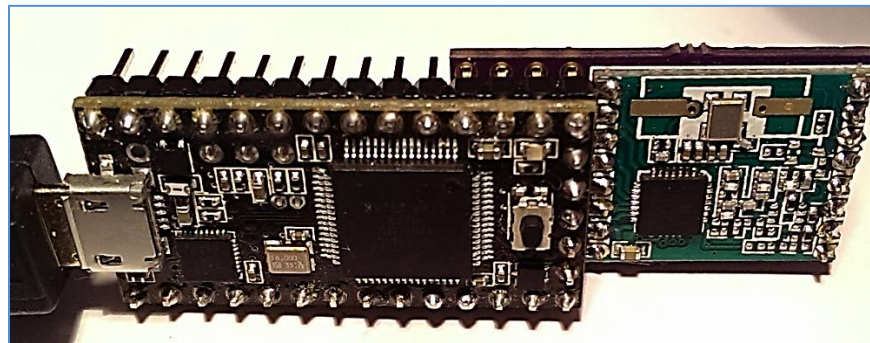
**RFM69 Pin-out and orientation, top view; radio photo superimposed**

Teensy 3 Signal Name	Teensy 3 Pin	RFM69 pin
Pin 11 alt. DOUT	11	MOSI
Pin 12 alt. DIN	12	MISO
Pin 13 ( LED) alt. SCK	13	SCK
N/C or alt. SCK (see note below)	14	
Pin 15 (SPI slave select)	15	NSS
Pin 16 (radio's DIO0) Interrupt	16	DIO0
GND	GND	GND
3.3V	3.3V	3.3V

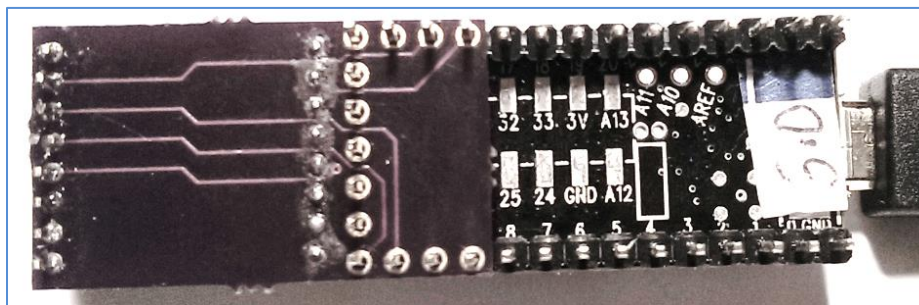
**Table, above: Adapter board printed circuit board connections**

The RFM69HW (H=high power) consumes, in transmit mode, more than the Teensy 3's specified 100mA maximum for the 3.3V from the ARM chip. This 3.3V is on two pins of the Teensy. Software could power-down the radio to allow some other peripheral on the Teensy 3 to share the 100mA. A bit risky though.

Note that pin 13, normally the blinky LED, is remapped to be the SPI port SCK. Some software remaps the SCK to pin 14 to leave pin 13 free to remain as the LED. The adapter would need a cut/jumper for this. Likely easier to add a new LED on some other pin.



Above: Note Orientation of radio! Teensy 3 with adapter and radio affixed, not soldered to Teensy y et. ready to test. Other methods to mate the boards include pin header and socket for stacking, etc.



Above: A bottom view of RFM69 adapter affixed to Teensy 3's bottom-side pins added in Step 2. In this photo, the adapter board is mated to the Teensy header pins but not soldered.

Option: One could use header pins in the RFM69 adapter board and sockets on the Teensy 3 to avoid soldering.

#### Step 4.

Solder the RFM69 to the adapter using the 2mm pitch/spaced holes. Caution again on proper orientation of the radio on the top side of the adapter board.

#### Step 5

Slide the adapter with RFM69 on to the Teensy 3 pins (bottom side of Teensy 3) as shown on previous page.

***The header pins fit snugly, perhaps with the adapter a bit ajar. Thus, the assembly can be software-tested before electing to solder the adapter to the Teensy 3 header pins.***

#### Step 6

Retest the Teensy 3 as in Step 1 to confirm there are no shorts or other flaws.

#### Tip:

The software can do pin remapping to move the SCK signal to pin 14 and preserve normal use of the LED on pin 13. The adapter board would need the SCK trace cut and jumper wire to do so. The code would be similar to:

```
#if defined (TEENSY3)
// First reassign pin 13 to Alt1 so that it is not SCK but the LED still works
CORE_PIN13_CONFIG = PORT_PCR_MUX(1); // Alt1 = PTC5. chip pin PTC5
// and then reassign pin 14 to SCK
CORE_PIN14_CONFIG = PORT_PCR_DSE | PORT_PCR_MUX(2); // Alt2=SPIO_SCK. chip pin PTD1
#endif
```

### Next: Diagnostic Software

## DIAGNOSTIC SOFTWARE - RFM69 and RFM22

The software shown on the next pages runs on a Teensy 3.1 ARM board (PJRC.com) and on an Atmel AVR Mega32 such as the RF69W mini-wireless board from Anarduino.com. The compiler/IDE can be Arduino 1.05 or Atmel Studio 6.1 with the Visual Micro plugin (both freeware) for the ARM and Teensy 3 boards, or other IDE. The code below can be downloaded from the Teensy 3 or RadioHead forum.

A common default SPI port slave select pin is 10 and is used in most freeware of Arduino heritage. As shown in the prior table, the adapter's PC board traces use pin 15 for slave select – to keep pin 10 free for other SPI devices. The other SPI port signals are on the same pins as most common software for SPI ports. *Given the above, freeware from the public domain needs a change to define the slave select as above.*

The adapter board used here is specific to the RFM69W or RFM69HW (high current). Many similar modules from HopeRF, Dorji, Nordic, Anarduino and others are supported by software such as RadioHead. Simple Software For Hardware Health-check

Teensy 3 libraries include SPI port support that is source code compatible with Arduino/AVR C/C++ code. Thus, either Arduino Atmel AVR or Teensy 3 can be the target for generic SPI based code.

The numerous configuration registers within the RFM69 are complex. The newcomer needs a known-good program and it must be in C/C++ and coded for Arduino (AVR) and thus can run too with the Teensy 3 and its Teensyduino library.

The recommendation is to use a simple application with the unmodified RadioHead library. First, a brief introduction to the library.

### RadioHead Library Overview

The AVR and Teensy 3 compatible RadioHead library provides the following:

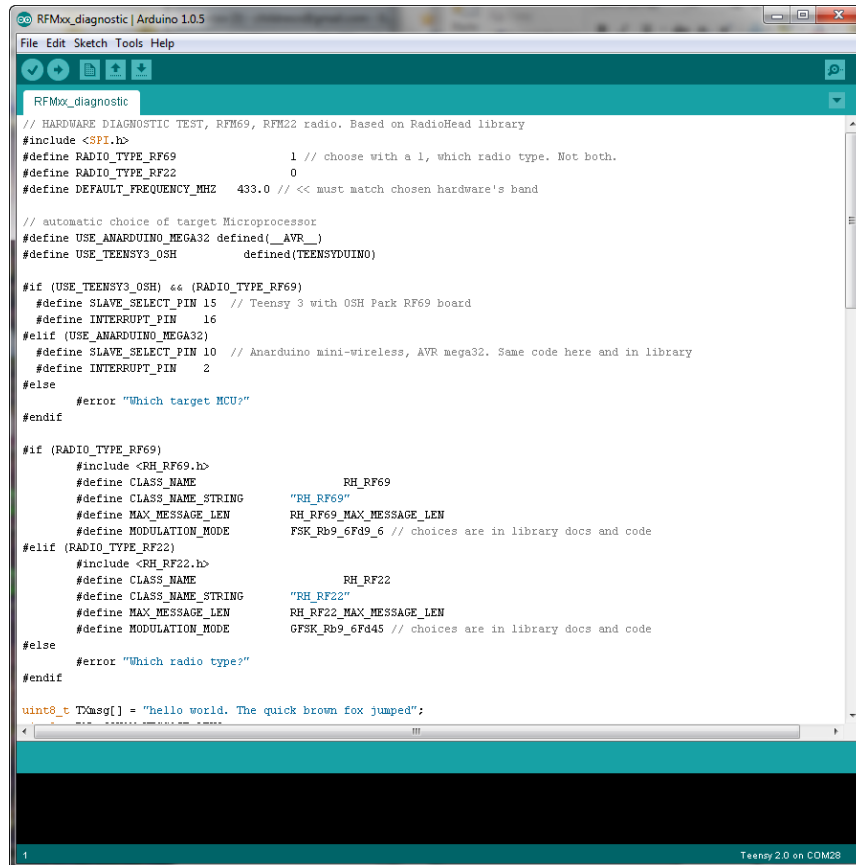
0. Radio hardware specific driver class. Here, this is for the RFM69.
1. Raw packet data with no destination address or error detection/correction (ACKs). The receiving node must have some assumption as to received packets' format.
2. Unreliable datagrams with addresses but no error correction (similar to IP's UDP)
3. Reliable datagrams with addresses and error correction ACKs and retries.
4. Routing among nodes where each has a pre-defined and unique static address
5. Ad-hoc Mesh routing. Self-forming, self-healing, re-routing as needed.
6. Other user developed protocols derived from the above, each of which is a base C++ class

Each node uses a link layer 8 bit address, 0-254. Address 255 is the broadcast address (no ACK) to reach any in-RF-range node that is in receiving mode. Nodes exchanging packets must each use the same network layer protocol as listed above. Each of the above is a C++ class. Number 0 is a radio. Example: to use protocol 3, an instance of protocol 0 is passed to the class constructor for 3. For more: Browse the mainpage and the links and class descriptions at

<http://www.airspayce.com/mikem/arduino/RadioHead/>

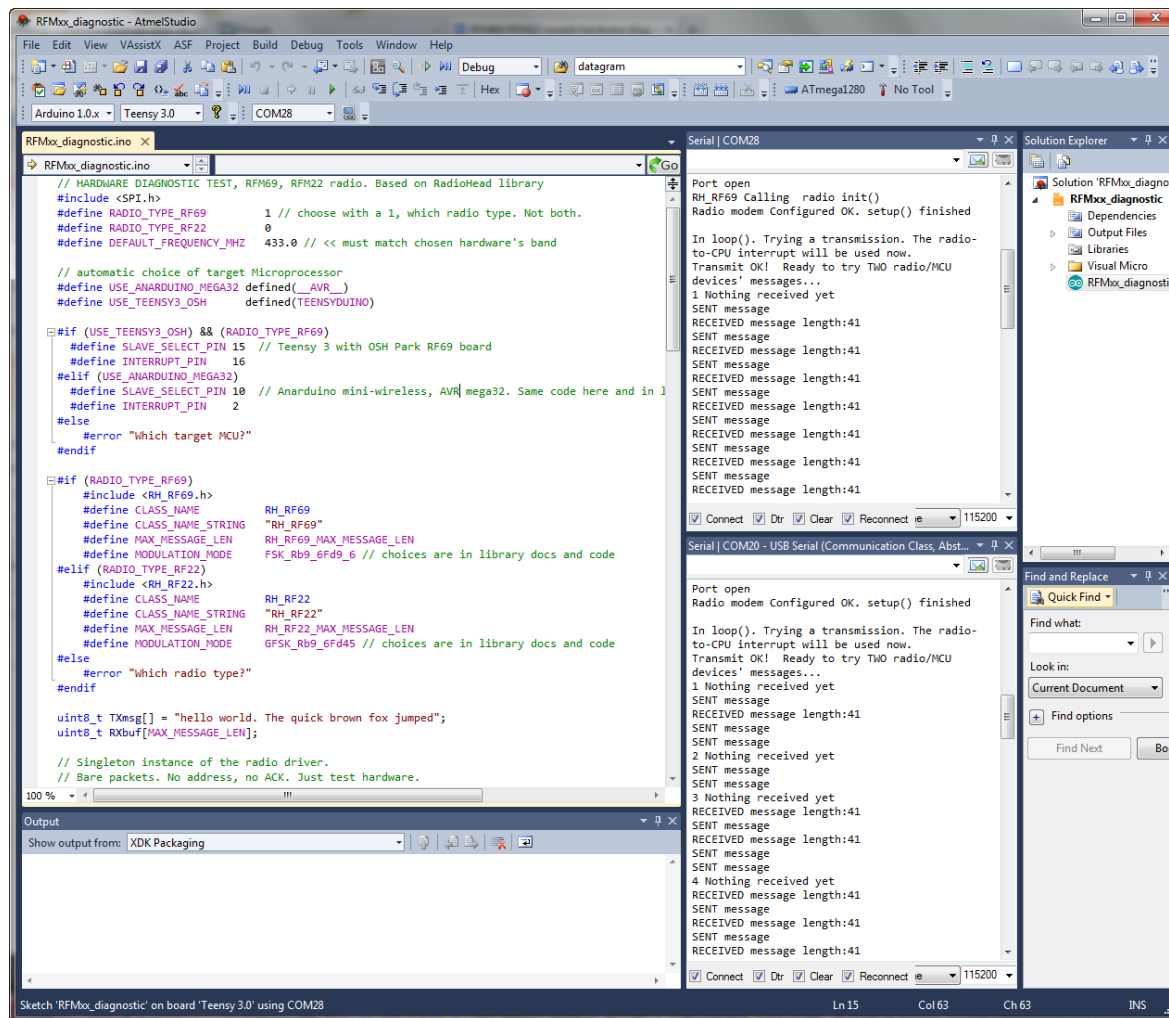
## Diagnostic Software

Code (below and in last pages of this document) runs on RFM69 + Teensy3 or a RFM69 + Atmel AVR Mega32. Both use the RadioHead library.

A screenshot of the Arduino IDE interface. The title bar reads 'RFM69\_diagnostic | Arduino 1.0.5'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for opening, saving, and running. The main text area contains C++ code for a hardware diagnostic test. The code includes comments and preprocessor directives for selecting between RFM69 and RFM22 radio modules and between Teensy3 and AVR Mega32 microcontrollers. It defines pins, interrupt pins, and message lengths. At the bottom, a string variable 'TxMsg' is initialized with the text 'hello world. The quick brown fox jumped:'. The status bar at the bottom right indicates 'Teensy 2.0 on COM23'.

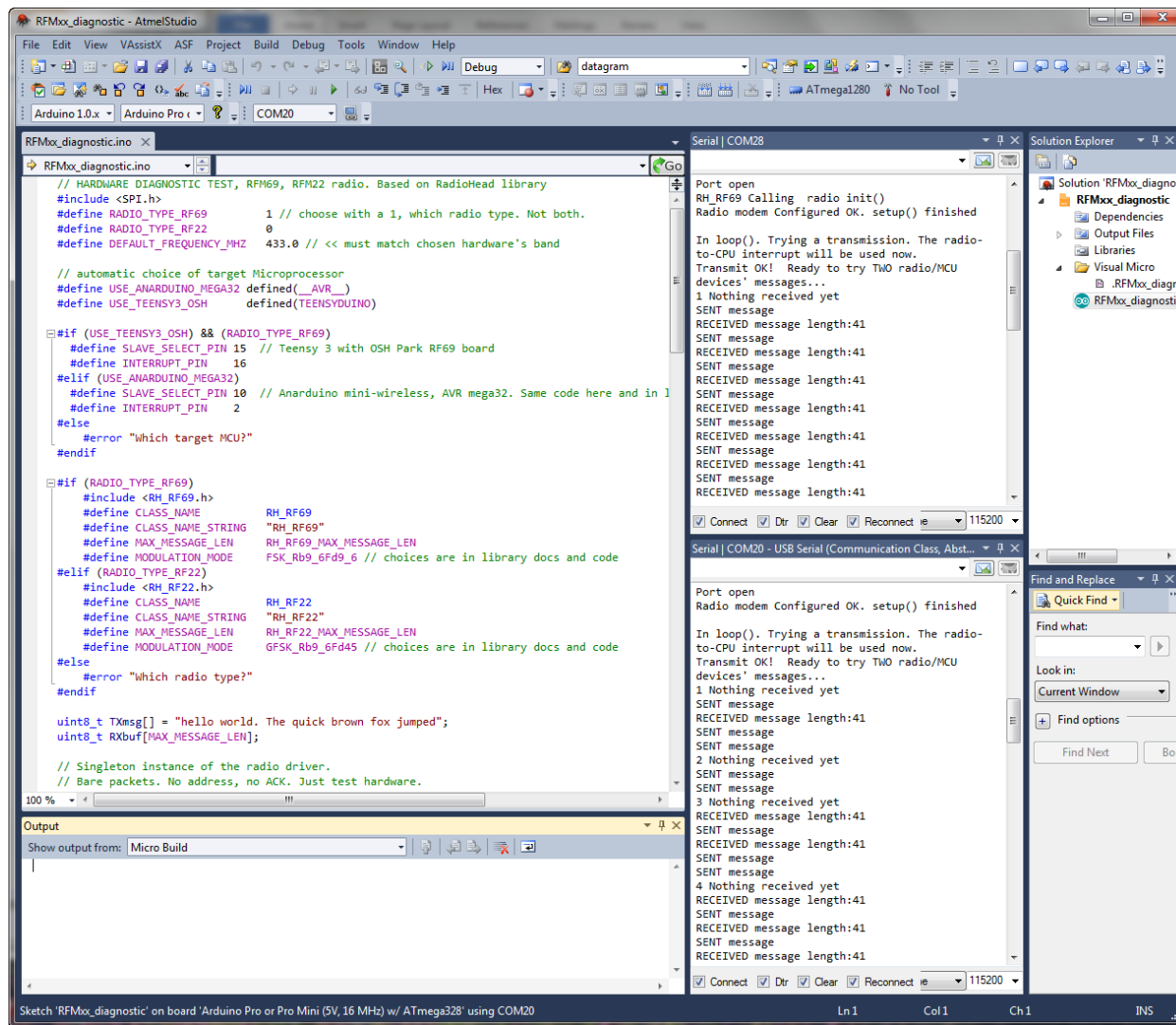
Above: Example of using the Arduino IDE for Teensy 3.1 as the target.





**Above:** Example of using Atmel Studio 6.1 with the Visual Micro plugin for AVR and Teensy 3. To the right of the code editor window are two vertically stack COM port terminal windows. These are displaying the serial/USB text messages from a Teensy 3 on COM20 and an AVR Mega32 (Anarduino miniwireless) on COM28, exchanging messages with the diagnostic.





**Above:** Same as previous page, but for the Anarduino miniwireless AVR Mega32 target.

```

// HARDWARE DIAGNOSTIC TEST, RFM69, RFM22 radio. Based on RadioHead library
#include <SPI.h>
#define RADIO_TYPE_RF69          1 // choose with a 1, which radio type. Not both.
#define RADIO_TYPE_RF22          0
#define DEFAULT_FREQUENCY_MHZ    433.0 // << must match chosen hardware's band

// automatic choice of target Microprocessor
#define USE_ANARDUINO_MEGA32 defined(__AVR__)
#define USE_TEENSY3_OSH        defined(TEENSYDUINO)

#if (USE_TEENSY3_OSH) && (RADIO_TYPE_RF69)
    #define SLAVE_SELECT_PIN 15 // Teensy 3 with OSH Park RF69 board
    #define INTERRUPT_PIN    16
#elif (USE_ANARDUINO_MEGA32)
    #define SLAVE_SELECT_PIN 10 // Anarduino mini-wireless, AVR mega32. Same code here and in library
    #define INTERRUPT_PIN    2
#else
    #error "Which target MCU?"
#endif

#if (RADIO_TYPE_RF69)
    #include <RH_RF69.h>
    #define CLASS_NAME          RH_RF69
    #define CLASS_NAME_STRING   "RH_RF69"
    #define MAX_MESSAGE_LEN     RH_RF69_MAX_MESSAGE_LEN
    #define MODULATION_MODE     FSK_Rb9_6Fd9_6 // choices are in library docs and code
#elif (RADIO_TYPE_RF22)
    #include <RH_RF22.h>
    #define CLASS_NAME          RH_RF22
    #define CLASS_NAME_STRING   "RH_RF22"
    #define MAX_MESSAGE_LEN     RH_RF22_MAX_MESSAGE_LEN
    #define MODULATION_MODE     GFSK_Rb9_6Fd45 // choices are in library docs and code
#else
    #error "Which radio type?"
#endif

uint8_t TXmsg[] = "hello world. The quick brown fox jumped";
uint8_t RXbuf[MAX_MESSAGE_LEN];

// Singleton instance of the radio driver.
// Bare packets. No address, no ACK. Just test hardware.
// below, "CLASS_NAME" is text given by the macro written above, based on radio type.
CLASS_NAME RHdriver(SLAVE_SELECT_PIN, INTERRUPT_PIN); // Create driver instance at compile-time.

// NOTE!!For this diagnostic only, there'll be no use of the higher level protocols. Just bare packets.

```

```

// The below would be used to extend the driver to support one of the protocols.
// #include <RHReliableDatagram.h> // from RadioHead library. It extends driver such as RH_RFxx.cpp
// RHReliableDatagram protocol(driver, myAddress); // pass radio driver class instance to constructor

////////////////////////////////////
void setup()
{
  Serial.begin(115200);
  delay(1000);
  while (!Serial) {}; // let host PC connect for serial
  Serial.print(CLASS_NAME_STRING);
  Serial.println(F(" Calling radio init()"));
  while (!RHdriver.init()) {
    Serial.println(F("driver.init failed. Check radio power, SS, SCK, MISO, MOSI wiring. Interrupt not needed at this
point"));
    delay(1000);
  }

  if (!RHdriver.setFrequency(DEFAULT_FREQUENCY_MHZ)) {
    Serial.println("setFrequency failed");
    while(1); // hang
  }
  if (!RHdriver.setModemConfig(CLASS_NAME::MODULATION_MODE)) {
    Serial.println("setModemConfig failed. Wrong freq. or modulation mode in software");
    while (1); // hang
  }
  Serial.println(F("Radio modem Configured OK. setup() finished"));
}

////////////////////////////////////
void loop()
{
  uint32_t lastTXtime = millis();
  uint32_t lastRXwarning = millis();
  uint8_t len;
  int pass = 0;

  Serial.println(F("\nIn loop(). Trying a transmission. The radio-to-CPU interrupt will be used now.));
  RHdriver.send(TXmsg, sizeof(TXmsg)+1); // send bare packet, C string w/null
  if (!RHdriver.waitPacketSent(500))
  {
    Serial.println(F("Timeout on transmit complete interrupt. Check Interrupt pin wiring"));
    while(1); // hang
  }
  Serial.println(F("Transmit OK! Ready to try TWO radio/MCU devices' messages..."));
}

```

```

delay(2000);

while (1) {
  if (!RHdriver.available()) // false if no message arrived yet
  {
    if ((millis() - lastRXwarning) > 1000)
    {
      Serial.print(++pass);
      lastRXwarning = millis();
      Serial.println(F(" Nothing received yet"));
    }
    if ( (millis() - lastTXtime) > 500)
    {
      RHdriver.send(TXmsg, sizeof(TXmsg)+1); // Send data now and then
      RHdriver.waitPacketSent(); // .available() above turns on receiver
      Serial.println("SENT message");
      lastTXtime = millis();
    }
  }
  else // have RX msg now...
  {
    len = sizeof(RXbuf); // must initialize to largest possible msg
    if (RHdriver.recv(RXbuf, &len)) // pass addr of len
    {
      Serial.print(F("RECEIVED message length:")); Serial.println(len);
      lastRXwarning = millis();
    }
  }
} // while loop
}

```