UKHAS Wiki

UK High Altitude Society

Linking an Arduino to a Radiometrix NTX2B Transmitter

Part 1 - Test Circuit and Test Code

Introduction

Getting your Arduino to transmit via the radio initially may seem daunting but its actually pretty simple. Please freely substitute the word “Arduino” for any microcontroller you wish to use. The example below works for 5V and 3.3V microcontrollers.

Please read this :

This may be the first bit of code you've come across with regards to creating a tracker. There is always the tempation to cut and paste it and away you go. And in isolation this may work. However should you then cut and paste further code without understanding what is going you will end up with an unworkable mess and something that is next to impossible for us to assist you with. Pretty please take the time to work out what the code below is doing, redo it yourself, break it, fix it most importantly understand it.

In this short tutorial I’ll give a step by step instructions on connecting a Radiometrix NTX2B [https://store.uputronics.com/index.php?route=product/product&path=61&product_id=60] to an Arduino and getting it to shift its transmission between two frequencies slightly giving two tones. In the second part I’ll demonstrate how to control this and use the circuit to transmit data at 50 or 300 baud using RTTY [http://en.wikipedia.org/wiki/Radioteletype].

The theory is as follows: you adjust the voltage on the NTX2B's TXD pin which adjusts its transmission frequency slightly. The difference in this frequency is called the shift. By doing this in a controlled fashion you can transmit 1's and 0's and therefore data. There are a number of methods to adjusting the voltage, connecting the TXD to a PWM on the Arduino, using a DAC or the most simple method discussed here using a voltage divider.

The NTX2B is a FM (Frequency Modulation) module intended to have a voltage applied to the TXD pin of between 0 and 3 volts. This voltage range changes the output frequency of the module by up to 6KHz. This means for each 1Hz change in frequency you need to change the voltage by 0.0005v (3 divided 6000) so to get a shift of 500hz you need to toggle the voltage applied to the TXD pin by 500×0.0005=0.25v.

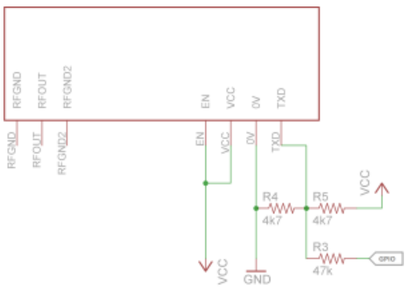
Getting the shift totally accurate isn't entirely essential but it should be kept in the 300-600Hz region.

Items Required

Radiometrix NTX2B
Arduino or similar microcontroller
1 x 47k resistor (Somewhere from 40k to 50k exact value isn't critical)
(For 3.3v somewhere round 32k)
2 x 4.7K resistor
Radio/SDR capable of receiving SSB at 434Mhz band.
PC with DL-FLDIGI installed.

Circuit Diagram

Firstly we build the following circuit (click it for larger) on the bread board :



The two 4k7 resistors, R4 and R5, sets the bias (centre) point of the two levels to be half the supply voltage. The GPIO pin then deviates the voltage from the centre point by an amount controlled by R3.

$$V_{cc} = 5V$$
$$GPIO_{low} = 0V$$
$$GPIO_{high} = V_{cc}$$
$$R4 = R5 = 4.7k$$
$$R3 = 47k$$
$$\text{NTX2 deviation} \approx 2000Hz/V$$

$$TXD_{low} = \frac{R4||R3}{R4||R3 + R5} V_{cc} = 2.38V$$

$$TXD_{high} = \frac{R4}{R5||R3 + R3} V_{cc} = 2.62V$$

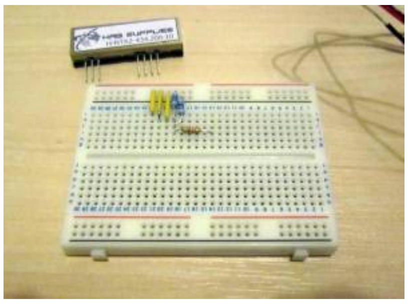
$$TXD_{high} - TXD_{low} = 0.24V$$

$$shift = 0.24 \times 2000 \approx 480Hz$$

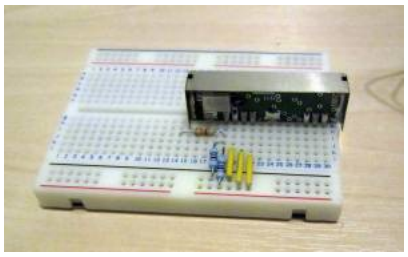
(Detailed explanation) [http://ukhas.org.uk/_detail/guides:ntx2_divider.jpg]

Build It

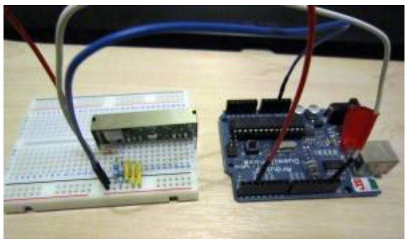
1/ Add the power and resistors as your board allows :



2/ Insert the NTX2. You can safely transmit with no antenna installed, no damage will occur to the module.



3/ From the Arduino connect 5V & GND and connect the pin marked RXD on the diagram to pin 13 on the Arduino.



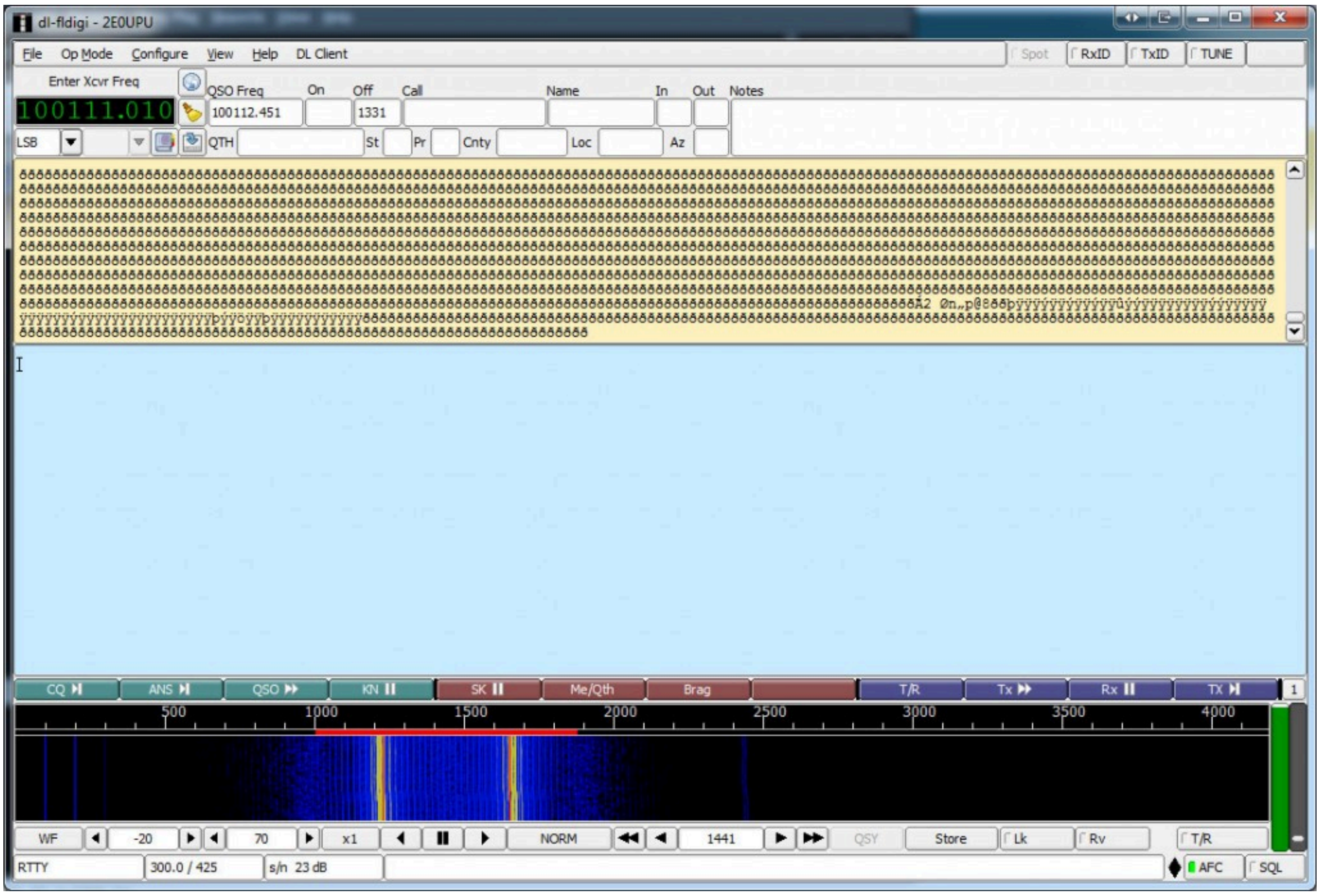
4/ Plug in the Arduino and load up the following code :

```
/*  NTX2 Radio Test Part 1

    Toggles the NTX2 high and low to generate a tone pattern to ascertain
    if the radio and associated circuitry is working.

    Created 2012 by M0UPU as part of a UKHAS Guide on linking NTX2 Modules to Arduino.

    http://ukhas.org.uk
*/

#define RADIOPIN 13

void setup()
{
pinMode(RADIOPIN,OUTPUT);
}

void loop()
{
digitalWrite(RADIOPIN, HIGH);
delay(100);
digitalWrite(RADIOPIN, LOW);
delay(100);
}
```

If you have used Arduino before this code will look familiar. This is because it is just the example code for Blinking the LED. In fact this will be currently blinking the LED in time to the radio changing frequency. All that is happening here is pin 13 is going from 0v (low) to 5v(high). This is reduced by the voltage divider on the NTX2's TXD pin (Note the NTX2 has an input resistor of 100k so this acts like another voltage divider in conjunction with the 47k/32k resistor).

5/ Tune in your radio to the frequency listed on your NTX2 module (either 434.075Mhz or 434.650Mhz) ensuring you have the mode set to USB. You should hear an alternating sequence of beeps. You may need to tune up or down a little, if you can't find it go back and check your circuit. Its most likely you'll find the transmission slighly below the frequency listed on the module.

6/ Now open up dl-fldigi and adjust the frequency and volume until you get a nice pair of lines on the water fall :



Enter Xcvr Freq 100111.010
QSO Freq 100112.451
Off 1331
LSB
-20
70
x1
1441
RTTY
300.0 / 425
s/n 23 dB

Note the distance between your lines is the shift and in this case is 425Hz, you can adjust this by playing with the value of the R3 resistor.

Congratulations you're transmitting using the most basic Arduino example code there is.

Part 2 - Making RTTY and transmitting data

Introduction

In part 1 of this article I discussed linking the NTX2 to the Arduino and getting a high and low tone out of it. In this article we go one step further and turn this into a transmission of data.

Using the circuit discussed in the previous article upload the following code will transmit a short sentence at 50 baud, 7 bits ASCII 2 stop bits. Also the code adds a CRC checksum at the end of the data string.

The DATASTRING is passed to a procedure called rtty_txtstring which takes care of transmitting the data by breaking it down into characters, then it transmits the individual bits of those characters. The key to getting the baud rate correct is the timing. Theoretically 50 baud should be 1/50th of a second = 20000µ seconds. Now for some reason the Arduino (at least the one I have) doesn't seem to be able to accurately count this time interval so the delay is achieved by doing one delay of 10000µS and one of 10150µS.

300 baud after some playing seemed to be stable around 3370µS delay (300 baud should be 1/300s = 3333µS). You can uncomment the relevant lines out as needed.

Demo Code To Transmit RTTY

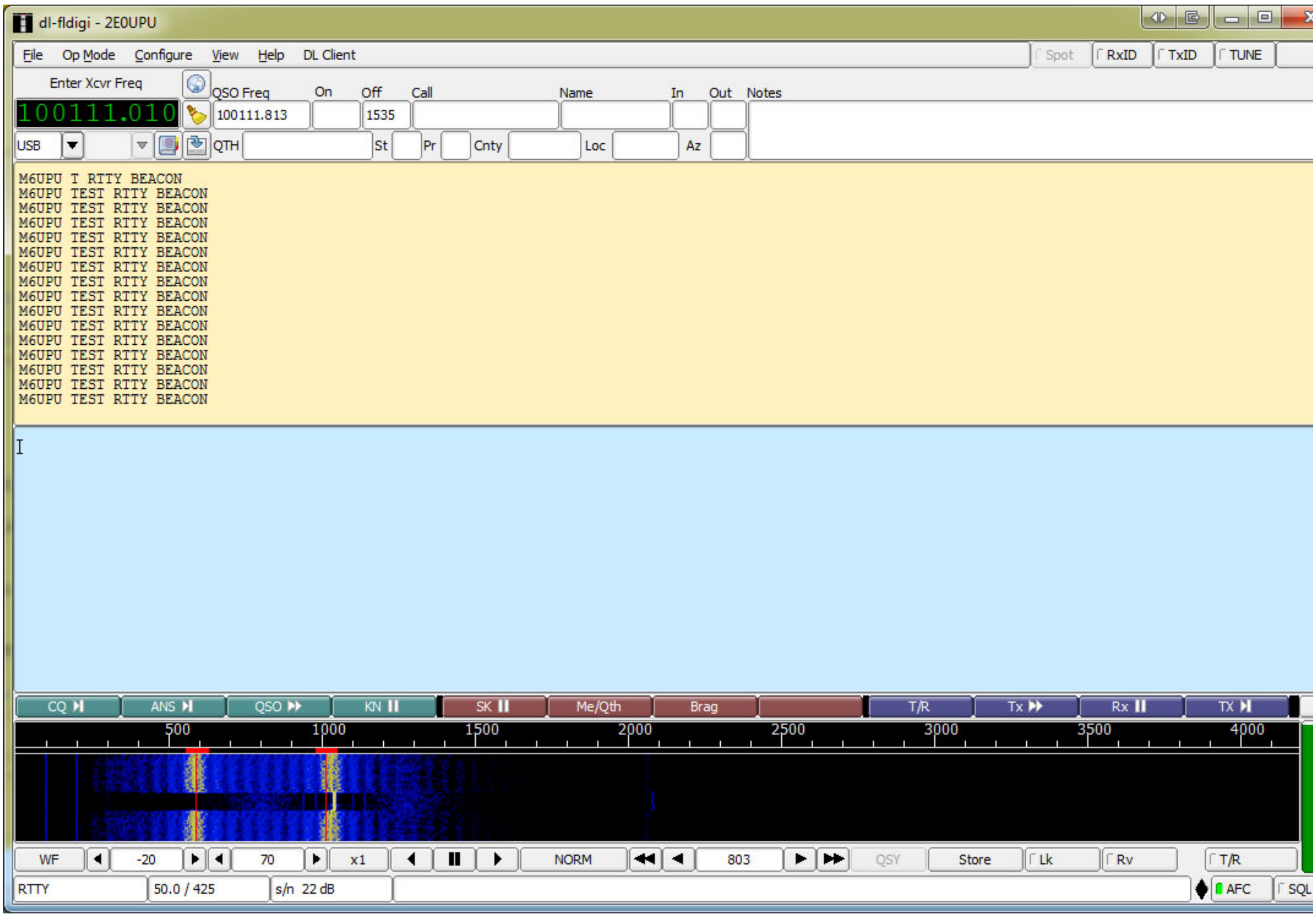
```
/*  NTX2 Radio Test Part 2

    Transmits data via RTTY with a checksum.

    Created 2012 by M0UPU as part of a UKHAS Guide on linking NTX2 Modules to Arduino.
    RTTY code from Rob Harrison Icarus Project.
    http://ukhas.org.uk
*/

#define RADIOPIN 13

#include <string.h>
#include <util/crc16.h>

char datastring[80];

void setup() {
  pinMode(RADIOPIN,OUTPUT);
}

void loop() {


  sprintf(datastring,"RTTY TEST BEACON RTTY TEST BEACON"); // Puts the text in the datastring
  unsigned int CHECKSUM = gps_CRC16_checksum(datastring);  // Calculates the checksum for this datastring
  char checksum_str[6];
  sprintf(checksum_str, "*%04X\n", CHECKSUM);
  strcat(datastring,checksum_str);

  rtty_txstring (datastring);
  delay(2000);
}
```

```
void rtty_txstring (char * string)
{

  /* Simple function to sent a char at a time to
        ** rtty_txbyte function.
        ** NB Each char is one byte (8 Bits)
        */

  char c;

  c = *string++;

  while ( c != '\0')
  {
    rtty_txbyte (c);
    c = *string++;
  }
}


void rtty_txbyte (char c)
{
  /* Simple function to sent each bit of a char to
        ** rtty_txbit function.
        ** NB The bits are sent Least Significant Bit first
        **
        ** All chars should be preceded with a 0 and
        ** proceded with a 1. 0 = Start bit; 1 = Stop bit
        **
        */

  int i;

  rtty_txbit (0); // Start bit

  // Send bits for for char LSB first

  for (i=0;i<7;i++) // Change this here 7 or 8 for ASCII-7 / ASCII-8
  {
    if (c & 1) rtty_txbit(1);

    else rtty_txbit(0);

    c = c >> 1;

  }

  rtty_txbit (1); // Stop bit
  rtty_txbit (1); // Stop bit
}

void rtty_txbit (int bit)
{
  if (bit)
  {
    // high
    digitalWrite(RADIOPIN, HIGH);
  }
  else
  {
    // low
    digitalWrite(RADIOPIN, LOW);

  }

  //                  delayMicroseconds(3370); // 300 baud
  delayMicroseconds(10000); // For 50 Baud uncomment this and the line below.
  delayMicroseconds(10150); // You can't do 20150 it just doesn't work as the
                            // largest value that will produce an accurate delay is 16383
                            // See : http://arduino.cc/en/Reference/DelayMicroseconds

}

uint16_t gps_CRC16_checksum (char *string)
{
  size_t i;
  uint16_t crc;
  uint8_t c;

  crc = 0xFFFF;

  // Calculate checksum ignoring the first two $s
  for (i = 2; i < strlen(string); i++)
  {
    c = string[i];
    crc = _crc_xmodem_update (crc, c);
  }

  return crc;
}
```

Upload the code and again open up DL-FLDIGI [http://ukhas.org.uk/projects:dl-fldigi] you should hear a wonderful sound of RTTY warbling away. Make sure DL-FLDIGI is set correctly by clicking Op Mode → RTTY → Custom.

Set the carrier shift to 425, baud to 50, 7 bits per character, no parity and 2 stop bits.

If everything is working ok you should get the data printing out :


dl-fldigi - 2E0UPU
File Op Mode Configure View Help DL Client
Spot RxID TxID TUNE
Enter Xcvr Freq
100111.010
QSO Freq 100111.813
On Off 1535 Call Name In Out Notes
USB QTH St Pr Cnty Loc Az
CQ ANS QSO KN SK Me/Qth Brag T/R Tx Rx TX
WF -20 70 x1 NORM 803 QSY Store Lk Rv T/R
RTTY 50.0 / 425 s/n 22 dB AFC SQL

```
M6UPU T RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
M6UPU TEST RTTY BEACON
```

Where to buy the NTX2

- Uputronics stock the NTX2B [https://store.uputronics.com/index.php?route=product/product&path=61&product_id=60]

Acknowledgements

Thanks to Rob Harrison for the RTTY code and to Dave Akerman for suggesting a more sensible one pin circuit to drive the NTX2B. Article authored by Anthony Stirk. Additional amendments by Keval, lz1dev, Costyn, Jcoxon.

Should you have any questions with this please come chat to us on IRC.

guides/linkingarduinotontx2.txt · Last modified: 2016/10/21 21:57 by upu