Radiometrix

Hartcran House, 231 Kenton Lane, Harrow, HA3 8RP, England

Tel: +44 (0) 20 8909 9595, Fax: +44 (0) 20 8909 2233

04 November 2008 **Application note 028**

Your data over a simple radio link

By Myk Dormer - Senior RF design engineer, Radiometrix

First published in Electronics World magazine. March 2007 issue

ISM band radio module manufacturers are offering higher and higher levels of user interface complexity, from elementary transparent RS232 'radio modems' up to the sophistication (and complexity) of Bluetooth or Zigbee protocol based radios. Many of these devices come with high-level software drivers, to link seamlessly into your ethernet or USB port.

But what if you are only trying to implement a simple remote control function ? Maybe a few user inputs, controlling a handful of relays, without the need to transfer megabytes of data, and without a powerful computer at either end of the link.

Despite what advertisers and industry pundits would have you believe, it is (still) possible to realise a reliable radio link with a pair of low cost radio modules and a handful of parts. Without megabytes of code, running on power hungry, complex, systems.

With only basic programming skills and a minimum of hardware or processing power, it is very easy to design your own "data over RF link protocol"

Firstly, if we examine the baseband the baseband (or 'data') interface of any simple FM wireless link pair, we find what is effectively an audio path through the radios, usually with limiter or squarer on the transmitter input and an 'average and compare' comparator on the receiver output, to permits direct connection to digital logic devices

It looks, from initial inspection of the data sheets, like it should be as simple as attaching the transmitter to a UART output, and the receiver to a UART input, and sending a byte or two through the link. If you try that, in most cases it will 'sort of' work, but the 'raw' radio path has a few unfortunate characteristics that need to be considered.

1. **The channel is noisy**: While this may not be evident during strong signal tests, as the range is increased the signal-to-noise of the received output degrades. (On the recovered digital output this manifests as increased jitter, and mid-bit 'dropouts'). In the presence of interference, such as other transmitters or environmental noise (ignition, lighting, electro-magnetic machines, or natural sources of noise like electrical storms) the wanted signal may be temporarily swamped entirely.
In the complete absence of a signal, a simple FM receiver outputs bandwidth limited white noise, which appears at the data output as a completely random datastream.

2. **The channel bandwidth is limited** (by a baseband filter in the transmitter, and by the receiver filters): The maker's data sheet will quote a maximum permitted data rate, but a non-time-critical application such as this one should be designed to operate at a much lower speed than this (to simplify the coding, and to maximise range for given S/N ratio)

3. **The channel is AC coupled** (despite the apparent logic level input and outputs): so there is a minimum frequency limit as well as a maximum. This imposes the need to keep the aggregate mark-space ratio of the burst close to 1:1, or the signal to noise will further degrade. In the worst case, a long enough string of continuous 'one' or 'zero' will cause the receiver's comparator reference to drift off and output spurious bit inversions

4. **The link has a defined, and sometimes quite long, set-up time**, required for the transmitter to reach full output and stabilise, and for the receiver to acquire an average for it's data recovery circuit. During this time no valid data can be passed, but it is necessary to provide a 'preamble' (a square wave sequence of consecutive high and low) to condition the data recovery circuit

From these characteristics it is possible to get an idea of what is required of the data format:

- Enough information (as framing, synchronising or addressing bits) needs to be sent in addition to the actual data or command sequence to allow the decoder to reliably discriminate between random noise in the absence of signal (or in the presence of interference) and the transmitted command burst.
- A bit level format is required which prevents long streams of continuous 'one' or 'zero' occurring, and which maintains an equal balance between high and low states (ie. has a high enough minimum frequency, and no DC offset). This format should also allow for a reasonably noise-tolerant decode

method.

- Data rate chosen must be as low as possible, within the system response time requirements. This will maximise the sensitivity of the link within
- The burst format must have a sufficiently long preamble (settling time) sequence preceding any decoded data.

A practical realisation:

It is possible to still use a hardware UART (rs232 type asynchronous interface) and fulfil all the requirements of the link, if a little care is taken.

The data is sent in a burst, or packet, comprising of:

[preamble]   [uart 'start' byte]   [framing bytes]   [data 'payload']   [checksum]

Characters must be sent continuously, start bit to stop bit without gaps (or the mark:space ratio will become unpredictable)

Assuming 1+8+1 (one start, 8 data, 1 stop) format, a sequence of 55h (ascii U) characters provides a square wave preamble. After the transmitter is turned on, a stream of these preamble bytes must be sent, until the 'tx-on' time spec. of the radio module used has been met. (This will usually be between 3 and 50mS, depending on the module)

An FFh byte must immediately follow the preamble, so the UART can frame on an identifiable start bit. To maintain DC balance, a 00h byte must follow the FFh. Both these bytes can form part of the decoded 'framing' sequence of the burst.

Then, to maintain mark:space balance, data can be sent by using only those characters with an equal number of ones and zeros in them. Of the 256 possible 8 bit codes, 70 contain 4 ones & 4 zeros. Omitting 0Fh, F0h, 3Ch and C3h (worst case 'four ones in a row' bit sequences) still leaves 66 usable codes per byte, which allows six bits of actual data to be coded into each transmitted byte.
(A full explanation of this method can be found here: http://www.radiometrix.co.uk/products/bimsheet.htm#rs232)

It is necessary to use a number of fixed value bytes preceding the actual data 'payload' as packet identification (or 'framing'), to allow the decoder to tell a valid data burst from random channel noise. (In my experience, to avoid false triggering of the decoder, at lease 3-4 bytes of framing data will be necessary.)

Additionally, one or more 'address' bytes may be added (to allow co-located operation of multiple systems, or polled access of multiple receivers by a single transmitter), and ideally, a checksum of some kind should be added to the data packet, as an extra precaution against spurious triggering.

I have described a very simple protocol here. There are many, far more sophisticated techniques in use throughout the industry. The method described does not even offer the best in S/N performance (as the edge triggered UART receiver is overly susceptible to data bit jitter and dropouts compared to a proper duration measuring biphase decoder) but it shows a workable technique, with a minimum of software effort and processing overheads.

Sometimes a simple approach is sufficient.

Radiometrix Ltd
Hartcran House
231 Kenton Lane
Harrow
Middlesex
HA3 8RP
ENGLAND
Tel: +44 (0) 20 8909 9595
Fax: +44 (0) 20 8909 2233
sales@radiometrix.com
www.radiometrix.com