

Introduction:

The HP series transmitters and receivers offer engineers a robust solution for making their products wireless. Because they are so simple to apply, it is easy to forget that they are radio frequency devices and subject to external influences not present in wired communications. In this application note, we will characterize these external influences (which can corrupt data in a wireless link). Once a full understanding of the possible sources of data corruption is achieved, we will focus on developing a simplex data transfer protocol framework to compensate for the inherent problems in a wireless environment.

There are many sources of interference that can cause data transmitted by a HP series transmitter to be corrupted. Some of the sources are within the control of the designer (e.g., power supply noise) and some are not (e.g. other licensed and unlicensed transmitters). Although the HP modules have been designed to be robust and reliable in the field, there is no guarantee that outside influences will not corrupt the data stream.

Therefore, we suggest that the designer implement some type of noise-tolerant protocol. The goal of the protocol is to synchronize communications between the transmitting and receiving ends, identify valid data packets, verify that the data packets are correct, and possibly even correct bad data in a packet.

To better understand the requirements for such a protocol, we should first examine all of the potential sources of data corruption that create the need for the protocol. To do this, we will develop a general model for the communications channel that the data must go through from the transmitter to the receiver. Using this model, we can account for all external and internal influences on the data stream.

Generic Communications Channel:

A communications channel is the path that data travels from the transmitter to the receiver and is made up of all of the components required to generate the data stream, encode it, transmit it (including the propagation path), receive it, decode it, and understand it. Figure 1 shows a generic model of a communications channel.

Data source – The data source can be anything. It could be an A/D converter reading

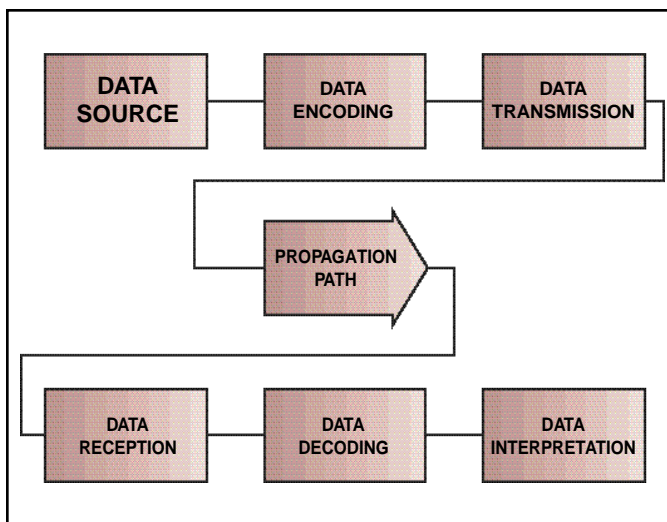


Figure 1

a temperature, a file on a computer hard disk, or a key press on a user input keypad. Data corruption at this stage in the communication channel is unlikely and can usually be traced to a bug in the hardware or software of the device in question.

Data encoding – The data coming from the data source is generally parallel in nature. The HP series transmitters require a serial data stream format. Therefore, some type of parallel to serial conversion must happen. This is usually accomplished with a UART (Universal Asynchronous Receiver and Transmitter)

although it is sometimes accomplished in software. Data corruption at this stage in the communications channel is unlikely and can usually be traced to a bug in the hardware or software of the device in question.

Data transmission – In the case of an HP series based wireless data link, the data transmission stage is an HP series transmitter. All HP series transmitters are fully tested at the factory. However, external factors such as power supply noise, improper modulation voltage levels, and improper antenna loading can corrupt the data stream. An HP transmitter, which is operated as recommended by its data sheet, should not contribute to data corruption.

Propagation path – The propagation path is the path that the radio waves take from the transmitter to the receiver. Data corruption is most likely to occur as the result of either in-band interference or desensitization from RF sources present in the propagation path. Additionally, conditions such as multi-path and fading can cause data corruption when the receiver is a given distance from the transmitter. All of these effects can cause continuous, erroneous data to be received.

Data reception – In the case of an HP series-based wireless data link, the data reception stage is an HP series receiver. All HP series receivers are fully tested at the factory to ensure that they function to all of the specifications set forth in the HP series receiver data manual. There are several conditions, though, that can cause data corruption in the receiver stage. The HP receiver's data output is squelched when the HP transmitter is not on. This is accomplished using the RSSI level compared to a factory-calibrated threshold. If another transmitter turns on with sufficient power, it can "desquelch" the receiver's output, causing the data output to become noisy. Furthermore, in-band interference and de-sensitization can cause the receiver to corrupt the data stream when it is a

distance away from the transmitter. And, lastly, the receiver can corrupt the data stream if the data source violates the minimum baud rate specification of the HP receiver, or if the very first byte of a data stream has bit 0 or bit 1 as a 0.

Data Decoding – Microprocessors and micro-controllers generally deal with data in their parallel form. Since the data coming from the HP receiver are serial, some form of serial-to-parallel conversion must be performed. Some form of UART usually accomplishes this. Data corruption at this stage in the communications channel is unlikely and can usually be traced to a hardware or software bug. When the data stream is corrupted prior to this stage in the communications channel, the errant bytes can sometimes be identified by a framing error.

Data Interpretation – Data interpretation is usually accomplished in software and involves doing something useful with the information that was received. Error detection and correction is usually implemented at this stage. Data corruption at this stage in the communications channel is unlikely and can usually be traced to a hardware or software bug.

In a wired application, the propagation path is a wire instead of free space and tends to introduce few if any errors into the data stream. This fundamental difference requires the need for some type of data transfer protocol. Protocol is defined in the next section.

What is a Protocol?

Every time two people communicate, there is always a potential for misunderstanding. In some cases, the risk is acceptable. Other times, such as during negotiations, misunderstanding can cause severe repercussions (such as war). When the accuracy of communications is vital, rules for communication are established prior to

negotiations to ensure that both sides are “speaking the same language”. These rules are referred to as *protocols*.

In a simplex data link, communications only go one direction: from the transmitter to the receiver. And the communications can be corrupted between the transmitter and receiver. A protocol is absolutely required in this environment to ensure that the receiver can “understand” the data from the transmitter and also determine if the data that was received was the actual data sent by the transmitter (e.g., are there any errors in the data stream?).

Goals of a Simplex Wireless Data Transfer Protocol

- Minimum overhead - a wireless data transfer protocol should be efficient. A protocol must add information to the main data. This information contains packet identification codes, error checking, etc.. The amount of information added should be the minimum amount required to achieve all of the goals of the wireless data transfer protocol.
- Reliable – a protocol is said to be reliable if it can separate good data from errant data. Reliability is usually achieved by embedding some form of error detection in the data stream. Parity, checksums, and CRCs are all forms of error detection codes.
- Robust – a protocol is said to be robust if it can correct errant data. Robustness is usually achieved by embedding forward error correction codes in the data stream. Forward error correction is a very involved subject in itself and will get little coverage in this applications note. For the purpose of this applications note, we will refer to forward error correction as any effort to correct errant data at the reception end by embedding redundant copies of the main

data in the data stream.

- Optimal radio performance – a wireless protocol should operate in a way that takes maximum advantage of the transmitter and receiver.

Packetization

A protocol will cut the main data into smaller pieces and wrap the data with special information needed by the protocol at the reception end. This process is called packetization.

At the reception end, the protocol strips the extra data from the packet, leaving only the original information. This process is called depacketization.

Start Codes and Noise

The first thing any protocol must be able to do is identify the difference between noise and valid data.

Noise appears as random bytes of information, with no obvious pattern. An ideal noise source has the ability to generate any combination of bytes with the same probability as any other combination of bytes. This property of noise makes it very difficult to find a combination of bytes to signify the start of a valid packet. Fortunately, in the real world, noise is rarely ideal.

Linx conducted extensive testing on the nature of noise generated by the HP series receivers under two conditions: desquelching and a white noise modulated carrier.

De-squelching occurs when RF energy is present at the antenna port of the receiver, which is greater in strength than the squelch threshold set at the Linx factory. When this happens, the receiver's data output appears to be noise.

In both cases, it was found that the combination of a 255 (or FF hex) followed by a 0 does not naturally occur in the noise.

The transmission protocol should begin a data packet with a 255 followed by a 0. The receiver protocol would then only accept packets that start with a 255 followed by a 0. Both start bytes **MUST** be received with errors, i.e., a valid start condition is not detected if a framing error occurs in either of the start code bytes.

Error Detection

Error detection is achieved by performing some type of analysis on the data prior to transmission and adding the results of this analysis to the data packet. Then, the same analysis is performed at the reception end and compared to the results embedded in the packet. If the two are different, then the packet is errant.

To better understand this process, let's work an example using the simplest form of error detection: parity. A parity check is accomplished by adding all of the 1's in a string of bits. If the number is even, then the parity bit is set.

Example 1:

In this example, we will calculate the parity of a byte that is to be transmitted. During transmission, one of the bits gets reversed. The receiver catches this through the parity check at the reception end.

Transmitter:

10101010

There are an even number of 1's, so the parity is set and the byte is transmitted as follows:

101010101

Receiver:

001010101

The last bit is not used in the parity calculation since it is the parity bit from the transmitter. Calculating parity on the received byte yields an odd number of 1's and the parity is 0. Since the receiver's parity calculation does not match the transmitter's calculation, the byte is determined to be errant and is thrown out.

The parity check is the easiest to implement, but is also the most unreliable. It can only catch an odd number of errors in the bit stream. If the number of errors is even, then the parity calculation will incorrectly indicate that the byte is good. Thus, there is 50% chance of the parity check catching an error (which is less than optimal).

Another form of error checking is the checksum. A checksum is calculated based on a series of bytes by adding the values of the bytes together and truncating the result to the desired bit length. For example:

4	data byte 1
109	data byte 2
65	data byte 3
<u>204</u>	<u>data byte 4</u>

126 8 bit checksum

A checksum will catch many more errors than the parity check. However, by simply transposing data bytes 2 and 3, the data packet becomes errant, but the checksum would provide the same result. The checksum only gives weight to the value of the bytes, not their order. Thus, errors of ordering cannot be caught with the checksum.

The most popular form of error checking is the CRC. The math to generate CRCs is complicated and beyond the scope of this applications note and will not be covered here. A CRC (cyclic redundancy check) is more reliable than the checksum because it gives weight to both the value of the data bytes and their position.

Error Correction

The goal of error correction is to embed redundant data in the packet at the transmitter end so that the receiver can correct the data if the error detection mechanism indicates that the data are errant.

Many forms of forward error correction have been developed over the years, including the ever-popular Hamming and Reed-Solomon codes.

Hamming codes use a modification of the parity check to detect and identify a single errant bit in a byte. They are used with great success in computer memory applications. In wireless data links, though, the noise tends to be bursting and will corrupt several concurrent bits. The Hamming codes cannot correct these type of errors and are therefore not suitable for use in a wireless data protocol.

Reed-Solomon codes have the capability of identifying and correcting strings of errant bits. They are used in everything from satellite transmission to CD players to compensate for interference and physical deformities (i.e. a scratch on a CD). Like CRCs, the Reed-Solomon codes are math-intensive and beyond the scope of this applications note.

A very simple method of forward error correction has been developed at Linx that is suitable for many wireless data links. The data are duplicated two times (for a total of 3 copies) in the packet at the transmission end. At the reception end, the first copy of the data in the packet is checked for errors. If there is an error, the two redundant copies of the data in the packet are used to generate one correct version of the data.

The correction is achieved by comparing the bits of each of the three copies of the data. If two or more bits are set, the corrected version

has that bit set:

0 0 0 0 1 0 1 1	copy 1 (errant byte)
1 0 1 0 1 0 1 0	copy 2
1 0 1 1 1 0 1 0	copy 3 (errant copy)

1 0 1 0 1 0 1 0	corrected byte

Once all of the bytes are corrected, they should be resubmitted to the error checking procedure to verify that the corrected bytes are valid. If they are not, then the data are uncorrectable. Otherwise the data can be used.

Putting It All Together

A simple protocol has been developed using all of the information provided in this applications note.

The transmitting protocol assembles the data to be transmitted into packet form as follows:

[START1] [START2] [PACKETTYPE] [DATA 0]
... [DATA n] [8 bit checksum]

START1 is a 255 (or FF hex). START2 is a 0.

"PACKETTYPE" indicates the type of packet being sent. In the current version of the protocol, there are two types of packets: corrected and uncorrected. A corrected packet contains three copies of each byte to support data correction. An uncorrected packet only contains one copy of the data. Uncorrected packets require less overhead and are less robust than corrected packets.

Error detection is achieved using a simple 8 bit checksum.

The receiving protocol waits for a valid start condition to be detected (i.e., START1 followed immediately by a START2 with no errors). Once a valid start condition is detected, the receiver determines the packet type and pulls the correct number of data bytes from the receive buffer. The data are checked. If the data are errant and the packet is a corrected packet, the receiver will attempt to correct the data. Otherwise, the packet is discarded.

When the transmitter has no packets to transmit, it can optionally transmit a user-definable idle character as a beacon to allow the receiving end to identify a transmitter on a given channel.

VERSION II demonstration software provided with the master evaluation kit (or downloadable from our website) demonstrates the HP series modules operating with this protocol. The full source code is included, which is written in ANSI C and can be ported to other platforms easily. The source code is well documented and will serve as a practical reference for applying the concepts discussed in this applications note.

First Byte Considerations

The HP receivers have shown a tendency in some conditions to incorrectly receive the first of a string of bytes if the bit 0 or bit 1 of the first byte is a 0. The most common time that a user will see this effect is during evaluation of the product.

If the engineer uses a terminal emulation program to evaluate the product and tests the operation by simply typing text on the transmitter end, the data may be corrupted. In this situation, the HP module is not operating properly because the minimum baud rate is not being met between key-presses. Although the individual bytes themselves are being transmitted at the baud rate set in the terminal emulation program, the spacing between the bytes (often greater than 500mSec) violates the minimum baud rate requirement of 300 baud (with a byte time of 33mSec).

The HP module will operate correctly every time if the first byte sent is a 255, regardless of the time from the last byte. Since this correlates to the start code requirements for noise performance, we always recommend that the first byte sent after a 33 mSec quiet period be a 255.