A Simple RF Data Link

Information and explanation to create a successful wireless connection

Summary

Sending data or information through wireless connection is incomparable with sending signals through such as a copper cable or a glass fibre cable. The main reason for this is because of the difference in properties of the physical layer.

The wireless transportation of raw data (which exists out of ‘0’s and ‘1’s only) is practically impossible. In order to create a wireless connection, which is close to being flawless, you need to be aware of a lot of qualities. This document describes the behaviour of the physical EM layer and offers a bunch of improvements for the wireless connection that can be implemented in the microcontroller. Doing this prevents disappointing test results.

The information provided in this document is merely used as an example. This example provides information and insights on how to produce a good, running system.

Properties

Most, if not all, wireless signals are created through emission of an electromagnetic pulse. Both radio signals and light are known kinds of EM rays. This type of physical layer acts differently than a copper or glass fibre cable.

A signal cable usually exists out of 2 conductors. These are separated from one another due to isolation that is placed in between the conductors. By creating a circuit and sending an electric current through the conductors, the system can start sending signals. The cable can be covered, to protect it, and the entire system makes for a virtually perfect connection. There are very few external factors capable of influencing the signal and there is almost a 100% guarantee it is going to get transported. That means that a copper cable has a very high availability. This availability is even higher when a glass fibre cable is used.

Due to the fact that the connection is wireless, there is no physical contact between the transmitter and the receiver. This means we cannot send an electric current. The medium we are using to send the signal is nothing but air surrounding us. However, even if it were to take place in a vacuum, the emission of the EM rays would continue being achievable. One of the compatible variables is the frequency on which the channel is formed. External factors are the cause of interferences and disable the medium of transmitting the EM rays. Depending on the frequency there is also the case of the damped response being far too high for a signal to be formed. The signal weakens during transmission from the transmitter to the receiver. The physical distance is too big or the transmitted signal is too weak.

However, there are other factors involved. The channel also gets used by other devices. One of the frequencies that is often used is 433.93 MHz. That frequency is a general license free channel used, for example, for opening car doors. Every time either you or your neighbour opens the car doors, a signal gets transmitted on that frequency. There are plenty of other disrupting factors. A lot of wireless sound equipment use that same frequency, which results in a continual interference. An availability of 95% on this frequency makes for quite a pleasant result. A wireless connection on such a general channel often results in a very low availability.

Not a whole lot that can be done about these kinds of interferences. It is unimaginable to try to protect the medium, for it cannot be shielded or covered. Yet this fact, that it is incapable of being protected, is actually our biggest advantage. It enables us to transmit a signal without having to place a cable underneath or in your house.

To create a reliable wireless connection we need to be aware of the qualities of the medium and our own devices. The biggest hurdle is the manufacturer’s demand concerning the range of the transmitter/receiver. This demand is only achievable in an ideal situation.

What is the right way of dealing with interferences?

After this somewhat negative talk, regarding the properties, it is time to look from the bright side and offer some solutions. Thankfully there are enough ways to create a reliable connection and to get rid of unexpected interferences.

We are using a system which is built for the wireless transportation of 4 bytes in a simplex transmission system, as an example. In this case we are assuming that the high frequency signal can be processed by the receiver and is able to turn that into a reasonably reliable connection (~95%).

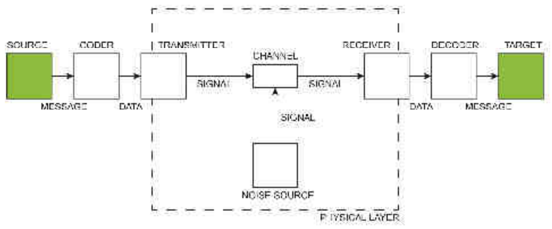

Figure 1; Simplex transmision systeem

Figure 1 shows how our simplex transmission system is built. The data is sent from left to right.

Due to the system's simplicity, we will not go into details about the generation of the EM rays, the modulation, reception of the EM rays and the demodulation (reconstruction). The general assumption is that after a working channel (= working transmitter and receiver on the same frequency) has been formed, data can be transported. We have provided recommended literature about modulation and the generation of EM rays in the bibliography. You can find the bibliography at the end of this document.

It comes down to this: The transmitter turns the data into an AC current, with a high frequency and with certain qualities (modulation), which makes it possible for the receiver to simply reconstruct the sent data. When the connection is sufficient, the antenna of the transmitter turns the AC current into EM rays. The receiver's antenna receives a small part of these rays (energy) and turns them into an AC current. The receiver can use this current to reconstruct the sent data.

When sending data using a wireless connection it is important to be aware of the fact that the receiver cannot sense a difference between a favourable and an unfavourable signal. After all, the physical layer is not covered. This means that we will have to apply an identification which enables the receiver to differentiate between a favourable and an unfavourable signal. In order to make this happen, we will have to add some qualities to the physical layer. By adding these resourceful qualities we can optimize the wireless connection.

Expanding the selectivity

As stated before, the sole selectivity the frequency has is a high frequency signal. However, this is not really useful because that channel is accessible to everyone.

Another way of avoiding interferences can be done by making use of time intervals, instead of sending a constant signal. In this case, transmitter is not always activated. Quite the contrary actually. One of the requirements necessary when using this channel (433.92 MHz), is to make sure the transmitter is only activated for a really short period. The transmitter will only be activated for the amount of time it takes to send the data. All the other (interfering) transmitters do this as well.

The complete avoidance of interferences when using a simplex connection (1 transmitter and 1 receiver) is not an easy task. To be able to use Collission Avoidance, the transmitter has to be able to evaluate the channel. The process can halt until the channel is completely unused, by making use of the data from the receiver.

The option to send data is nearly always available. This can be done o.a. by making use of the Brute Force method. This method includes the forcing of the signal, regardless of whether the channel is being used or not. By sending the data several times, instead of once, the availability increases. Even if the data is not received the first time around, it will definitely be received during the second or third transmission. This states that we are aware of situations in which the data cannot be received and that we might create interferences for other users of this channel. It is up to you, the user, to make sure all the information gets received. By activating the connection for the shortest period there is a higher chance of receiving all the information in one go and creating fewer interferences for other users.

The Packet

Our next step is to increase the selectivity: by adding information which enables the receiver to differentiate between all the sent data. Once added, our receiver will only receive information sent by our transmitter. In order to make this happen, we will wrap the data in a Frame of Packet, which has its own unique qualities.

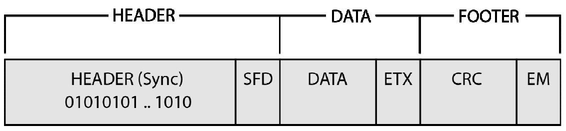

Figure 2, The packets lay-out

A single packet consists out of a header, the data itself and a footer. The header exists out of information which has been added for the sole purpose of synchronizing the UART in the microcontroller. The header very important when using a wireless connection. Without it, it would be practically impossible to transmit accurate data. The UART cannot synchronize when it has access to the data only. Without the header, the first few bytes will most likely be missed, which is unfavourable when you are trying to send a mere 4 bytes. You can solve this problem by implementing a header that consists out of at least 8 bytes.

The length of the header changes depending on how much time it takes for UART to synchronize. Sending data with a data rate of 2,4 kbit/s will most likely have a shorter header than when a data rate is 9,6 kbit/s is used. In theory, the header of the data, that has a data rate of 9,6 kbit/s, should be four times the length of the data with the data rate of 2,4 kbit/s. In reality, it is up to you to decide the length of the header. Choosing the appropriate length of the header can be done by running a couple of tests. First of all, test to see what the minimum length is keep the connection going. Secondly, multiply this length times two. If the system barely, but still, works when using 6 bytes; the length of the header has to be 12 bytes.

The end of the header is declared with a separating character, after which the data follows. The end of the data is also declared with a separating character. However, if the data has a stable length, you do not need to use such a character. The footer includes a final set of bits and possible a code; the result of a calculation with locks in the data.

The selectivity has increased by adding unique qualities to every bit of the packet's information. In turn this makes it virtually impossible for the receiving microcontroller to recognize a different (a signal you didn't sent) signal as the favourable signal.

The UART

The packet gets transmitted by sending a series of bytes (8 bits) with the UART. A Universal Asynchronous Receiver/Transmitter transmits the sets of 8 bits, one for every byte, using a steady signal. A singular start and one (or two) stop bit(s) are added to each of the sets:

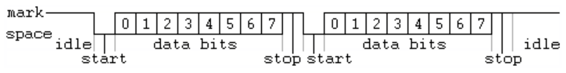

Figure 3, Output signaal of the UART

The packet is ready to be created due to the implementation of the bytes (characters). In order to make this happen, we use the ASCII (American Standard Code for Information Interexchange) code.

A good example of a packet is this one:

SYN-SYN-SYN-SYN-SYN-SYN-SYN-SYN-STX-DATA1-DATA2-DATA3-DATA4-ETX-EM

Something that is worth noticing is that, in order to transport 4 bytes (DATA1 to DATA4), a total of 15 bytes (150 bits) need to be transmitted!

The transmitter should be activated approximately 50 ms before sending the first byte. The transmitter uses this time to stabilize on the right output power and frequency. The receiver uses this time in order to respond and to switch the squelch circuit. To prevent serious problems from occurring, this amount of time should not be altered! A whopping 480 bits can be send in 50 ms, with a data rate of 9,6 kbit/s. We can transmit our packet thrice in that amount of time!

It might also be a good idea, however this is not compulsory, to keep the transmitter activated for another 5ms after having sent the final byte. It is often not possible to indicate, in the software, when the last bit has been sent. That is why we recommend the deactivation of the transmitter to take place 5ms later. By doing this you are making sure last byte is fully sent while the transmitter is still activated.

Concluding, the transmission of 4 bytes of raw data takes about 71 ms, with a data rate of 9,6 kbit/s (50 + (150 x 9600^-1) + 5). With a data rate of 2,4 kbit/s, this period becomes about 118 ms, which is still short period of time. After all, the shorter the packet the bigger the chance it gets received as a whole, by the receiver.

Extra protection

To protect the raw data even better against possible errors we can add redundancy. Just like repeatedly sending the complete packet it is also possible to repeatedly send the data only. Our assumption in this case is that the disrupting signals will not last for too long. The header and the footer remain the same. Note that this increases the length of the packet and simultaneously the amount of time it takes to send it.

Another kind of redundancy is an encryption. This encryption's ability is to detect and correct bit errors. Due to the simplex connection, the ability to only detect mistakes is not that useful. After all, there is no way of telling if the transmitter the packet needs to send again.

A proper yet complex method of protecting data is by adding a mechanism that detects and corrects errors. This could be done with e.g. a FEC or BCH code. This kind of protection encrypts the data and adds extra bits to the packet.

It is but a question whether it is worth implementing these software protection methods, especially since errors and misuses of equipment, by other parties, do not occur on a regular basis. All these methods might be a bit too much when using, for example, a simple Domotica system that is meant to turn a lamp on and off. In that case, it is best to choose a simpler protection method.

Software

It is up to you to create a beautiful system by writing the correct software. To make sure the system is universally compatible, a rather limited interface is available and virtually every microcontroller can be used. The amount of variables makes it impossible for us to give examples of software code.

What we can do is suggest a global way of programming. In this example we will use the programming of the receiver.

The first step is to choose the right kind of microcontroller. Pick a microcontroller which has UART hardware implemented into it. The UART has to be activated and configured to the correct BAUDRATE and assembly. This configuration has to be the same for both the transmitter and the receiver.

The software works easiest on interrupts. The UART transmits an interrupts after it has received a byte. The ISR puts the byte on a free spot in a list and increases a pointer. When a byte has not been received 5 times in a row (a time-out), the ISR will send a signal to the main loop to indicate a packet has been received. The main loop will decode and process the data afterwards.

The same can be achieved without the use of interrupts. The UART puts the received byte on a spot and puts a 'byte received' message on a fixed place. This enables the possibility for the main loop to easily detect whether a new byte has been received. The polling frequency of the main loop has to be higher than the set BAUDRATE.

After this the main loop can start evaluating the series of received bytes. By adding a unique code to the header of the packet, the main loop can easily decide if the message is meant for this particular/our receiver.

Addressing (when multiple receiving devices are activated) is usually not fixed in the header. A better suggestion for fixing this is to use (an) extra databyte(s).

Protection

Author:

IDEETRON
ELECTRONICS PROJECTS

Ideetron b.v.
Tel: +31 (0) 343 769 094
e-mail: info@ideetron.nl
www.ideetron.nl