An Encoding Scheme for RFID Transmission

Craig Pataky

December 5, 2008

Just as bar code technology revolutionized the product identification industry in the 1990's, radio frequency identification (RFID) is poised to perform the same feat today. Semiconductors are now available that can produce tags and readers at such low cost as to make radio technology nearly as ubiquitous as printed ink technology.

Basic to the operation of RFID is a small radio transmitter that periodically broadcasts its serial number to a receiver. All this sounds simple so far.

As I research the offerings of DataLogic, Motorola, ActiveWave, and Omron, it is clear that seemingly complex protocols are already in existence. Larger companies seem to adhere to interoperable ISO standards regarding transmission protocols, where small to midsize companies remain at liberty to develop and deploy proprietary protocols.

There is apparently no hard and fast rule that an RFID solution must operate one way or another. If a scheme can get data from tag to reader and reasonably ensure some data integrity, then that's an RFID system. There are no RFID referees. There are no RFID sheriffs. You can really do whatever you want to do.

Keep it Simple

I like simplicity. Simplicity works the first time, every time. As I examine the available solutions, I've got to cast my vote for the fewest components, the least software, and the most readily understandable.

One such system might involve a 315mhz transmitter and receiver. As a basis of discussion, let's imagine a case where the transmitter is either ON or OFF. If the receiver "hears" that the transmitter is on, the output pin of the receiver goes HIGH. If the receiver does not "hear" the transmitter, the output pin of the receiver goes LOW.

Transmitter	Receiver Output
ON	HIGH
OFF	LOW

Table 1: Relationship of transmitter status to receiver output.

Given that we now have a way to control the logic state of receiver output, it follows that we can encode a pulse train on that line to represent a unique tag identifier. As with everything else in electronics, we start with a basic representation of bits 0 or 1.

A simple way to transmit 0 might be to key the transmitter for a short time and then abruptly turn the transmitter off. Similarly, transmitting a 1 might involve keying the transmitter for a noticeably longer time and abruptly turning the transmitter off.

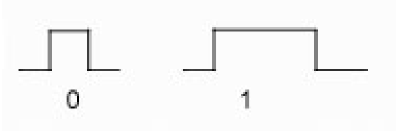

Fig 1: A zero is a short pulse, a 1 is a longer pulse.

When the receiver "hears" the tag (or not) and sets its output to either a HIGH or LOW state, we can conceive the binary building blocks of a more intricate data representation.

Tag Representation

As we string a series of 1 or 0 bits together, we can form bytes (8 bits) or words (16 bits) and still more multiples of these basic elements can represent a complete RFID tag.

For a tag to accurately report its position, the tag must provide three essential ingredients:

1) Activator ID – The method that prompted the tag to transmit.
2) Facility Code – The building or company that owns the tag.
3) Tag ID – The serial number of the tag itself.

Arbitrarily speaking, I couldn't imagine a need for more than 256 Facility Codes or 256 Activator IDs. We could then represent each of these elements as 8-bit bytes. There might very well be several thousand tags in a given inventory, but probably not more than 65535. We can therefore safely represent a Tag ID using a 16-bit word. This would give us an overall tag makeup as follows: [BYTE][BYTE][WORD]

A Close Look

Now that we have a good formula, let's examine how such data might be represented at the basic level. To start, let's assume we have a tag fully represented as: 25202548511

We begin with the 8-bit Activator ID:

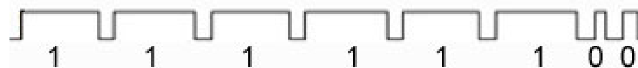

Fig 2: The Activator ID of 252 (FCh) as represented.

Next we consider the 8-bit Facility Code:

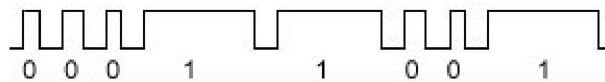

Fig 3: The Facility Code 025 (19h) is represented.

After the Activator and Facility code, the tag's own 16-bit Serial Number is sent:

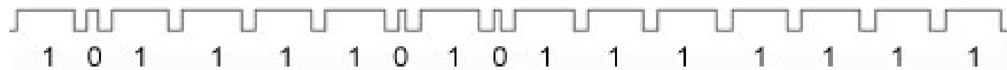

Fig 4: The Serial Number 48511 (BD7Fh) is represented.

Data Integrity

As with most protocols, it is desirable to append a CRC to the data so that the receiver can verify data accuracy. The CRC is transmitted by a tag the same way as any other byte--A short pulse represents a 0 bit and a longer pulse represents a 1 bit. Building on our previous example, the CRC for our theoretical tag is represented as follows:

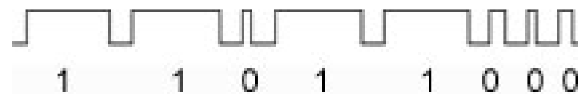

Fig 5: The CRC 216 (D8h) is represented.

As it so happens, I have a simple 8-bit CRC algorithm laying around that I learned from Dallas Semiconductor back in the mid 1990's. In this case, we run all four bytes of our tag data through the CRC engine, logically NOT the result, and we should have the accurate integrity check. If the value we generate using this method does not match that sent by the tag, we simply ignore the tag.

The actual function used to calculate the CRC is as follows:

```
BYTE CRC_Byte(BYTE Seed,BYTE Data)
{
 int j;

 for (j=0;j<8;j++)
  {
   if (((Data^Seed)&1)!=0)
    {
     Seed^0x18;
     Seed=Seed>>1;
     Seed|=0x80;
    }
   else
    {
     Seed=Seed>>1;
    }
   Data=Data>>1;
  }
 return(Seed);
}
```

In practice, ur CRC function could be used in the following manner:

```
kalk=0; //Seed the calculated CRC with 0
kalk=CRC_Byte(kalk,activator);   //252
kalk=CRC_Byte(kalk,facility);    //025
kalk=CRC_Byte(kalk,tag_hi byte); //BDh
kalk=CRC_Byte(kalk,tag_lo byte); //7Fh
kalk=~kalk; //Invert the calculated CRC

if (((BYTE)kalk)!=((BYTE) tag_crc)) return(FALSE);   //CRC bad!
else return(TRUE); //CRC good!
```

How long is long? How short is short?
Thus far we've been rather generic in our explanation. It's all fine and well to discuss relative short and long pulse widths, but when the rubber hits the road we really need something more specific. After all, are we talking about milliseconds or microseconds? That's a big difference.

Preambles and Postambles
Considering the full pulse train of an RFID tag even further, I can imagine the tag data framed by a preamble and a postamble.

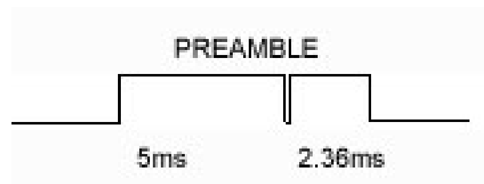

Fig 6: The preamble has two parts. One to break squelch, the other to calculate timing.

The preamble is broken into two parts. The first is a 5ms pulse that I assume is used to break receiver squelch and get the receiving microprocessor's attention. The second pulse is about 2.4ms and appears to be used for timing. If I divide 2.4ms by 3, I get 800 microseconds. I believe that this is the best way to discriminate between wide and narrow pulses:

If a pulse is longer than the timing pulse divided by three, it is a 1.
AND
If a pulse is shorter than the timing pulse divided by three, it is a 0.

Practical experience shows that a long pulse is about 1.4 milliseconds, where a short pulse is about 160 microseconds. The gap between pulses is about 280 microseconds.

Regarding the postamble, it could be just a single pulse of about 700 microseconds at the end of the train. It does not serve any practical purpose except to signify an official end to the tag data.

Putting It All Together
The total pulse train including preamble, tag information, and postamble might typically be around 50 milliseconds. An illustrative example of a complete expression is as follows:

Fig 7: The complete RFID pulse train.

Conclusion
By now we've examined all elements that could make up a perfectly useable RFID signaling method. Timing information can be encoded in the preamble, extensive tag data can be represented by a series of short and long pulses, integrity is ensured by use of a CRC, and a postamble can be used to signify the end of a pulse train.

I am sure there are other methods to encode a tag, but this one seems simple. I like it, and am sticking with it.

###