434MHz RF Protocol Descriptions for Wireless Weather Sensors

Oregon Scientific, AcuRite, Ambient Weather

October 2015

Much of the information here has been gathered from various postings on the internet. This document summarizes what has been already made public and adds a little more to the pile of knowledge.

The description is broken into two parts or layers which are named in accordance with the standard OSI model for network communications. While this may not be a perfect use of the model, it serves well enough for this purpose.

There is one final introductory point. This document uses the word "protocol" to refer to the overall collection of the two layers (physical and data link). Again, perhaps not in agreement with other uses of the word "protocol", but that's how it is used here.

The Physical Layer

This first section describes how bits are encoded into the RF signal. The quick summary is that all sensors listed here use on-off keying -- in other words the RF signal is either on full or completely off. The on/off state is varied over time to send the data...it is a lot like the old Morse code.

Oregon Scientific Protocol Versions 1.0, 2.1, 3.0

At the physical layer, RF transmissions use on-off-keying (OOK) with Manchester coding on a carrier frequency of 433.92MHz. See the Wikipedia entry on Manchester coding for more information or consult a basic text on digital RF communications.

All OS protocol versions use the “normal” polarity definition of Manchester coding. This convention requires that a zero bit be represented by an off-to-on transition of the RF signal at the middle of a clock period. Another way to describe this is that the bit value is equal to the RF signal state *before* the transition.

By definition, RF transitions must occur in the middle of each clock period. In this document, that point will be designated by an integer number. The boundary between two clock periods will therefore be equal to an integer plus one half.

OS version 1.0 sensors transmit bits with a clock rate of approximately 342Hz, while version 2.1 and 3.0 sensors use a bit rate of 1024Hz. In all version 2.1 and 3.0 sensors measured to date, this rate does not vary by more than a few tenths of a Hertz.

Sample recordings of RF messages are shown in the appendix at the end of this document.

OS Version 2.1 Message Formatting

For version 2.1 sensors only, each data bit is actually sent four times. This is accomplished by first sending each data bit twice (inverted the first time), doubling the size of the original message. A one bit is sent as a “01” sequence and a zero bit is sent as “10”. Secondly, the entire message is repeated once. Some sensors will insert a gap (about 10.9 msec for the THGR122NX) between the two repetitions, while others (e.g. UVR128) have no gap at all.

For an example of this, consider the message “111101010111” as it would be sent by a version 2.1 sensor. First an inverted copy of the message is created, and then interleaved with the original message, taking the inverted bit first.

```
Original Message:  1  1  1  1  0  1  0  1  0  1  1  1
Inverted Message: 0  0  0  0  1  0  1  0  1  0  0  0
Transmitted Bits: 01 01 01 01 10 01 10 01 10 01 01 01
```

When decoding a version 2.1 message, only every other bit need be used (and possibly inverted, depending on whether the first or second bit is kept). If the second bit in each bit pair is kept, no inversion is required.

It should be apparent for version 2.1 messages now, that one can assume the opposite polarity for Manchester coding (e.g. a zero bit is represented by an on-to-off transition in the RF signal) – this only changes which of the two interleaved bit streams is considered to be inverted.

OS Version 1.0 Message Formatting

These sensors also repeat each message once, but do not repeat each bit.

AcuRite VN1TX Integrated Sensor Suite Formatting

At the physical layer, this sensor uses OOK combined with pulse-width modulation (PWM). A one bit is transmitted with a long RF pulse and a zero bit with a short RF pulse. All RF off-to-on transitions occur on a regular time interval of approximately 620 microseconds or at a rate of about 1600Hz.

AcuRite 00592TXR Tower

OOK at the physical layer, identical to the VN1TX.

Ambient Weather WH2C

This sensor uses PWM like the VN1TX, but a one bit is coded as a short pulse and a zero bit as a long pulse. The preamble consists of eight one bits with no special timing compared to the rest of the transmission. Pulse widths are either 500us or 1500us. All off periods are 1000us.

Ambient Weather F007TH

Manchester coding is used at the physical layer by this sensor. Clock rate is 1024Hz within a very small window of variation. The preamble contains a total of 13 bits; the first 11 are ones, followed by a 01 sequence. The next bit begins the data frame.

SL-109H and AcuRite Sensor Formatting

These sensors, while still using OOK for modulation, use a completely different encoding for bits. Each RF pulse is always 500µsec long, and the bit value is encoded in the time spacing between pulses. This document will refer to this modulation format as "pulse-spacing modulation" or "PSM" in this document.

In addition to the OS SL-109H, AcuRite models known to use PSM are 00955, 00964TX and 00606TX.

With all of these sensors, each message is repeated multiple times -- four for the SL-109H, about ten or so for the AcuRite 00955 and six for the AcuRite 00606TX.

For the purposes of this document, pulse spacing is defined as the amount of time for which the RF signal is "off" between pulses. Using this definition, a "zero" bit is indicated by a pulse spacing of 2 msec and a "one" bit by a spacing of 4 msec. The end of the message is indicated by an RF off period of 9 msec. Following this long "off" period, another repetition of the message may follow, or the RF signal will remain off if there are no more repetitions to follow.

RF Pulse Widths

The duration of Manchester-coded RF pulses is exactly either ½ or 1 data clock period. However, the pulse widths seen in actual practice will vary from these ideals. There are two major reasons for this. First, the sensor itself may not be sending RF pulses with the ideal widths. Secondly, the receiver will often modify the width of received pulses as an artifact of its design.

A significant design challenge for receiver firmware is to be tolerant of these pulse width variations while still accurately decoding signals from the desired set of sensors.

At a typical receiver, OS version 2.1 and 3.0 signals exhibit shortened RF pulse widths (often) by truncating the end of the pulse, not the start of the pulse). As a result, RF transitions do not occur on exactly regular time boundaries and may be displaced in time from data clock edges. Pulses are shortened by about 138us for v3.0 sensors and 93us in v2.1.

Version 1.0 sensors have the RF pulses lengthened instead of shortened.

The Data Link Layer

This next layer up consists of "frames" which are groups of bits transmitted as a whole, unbroken unit. Although the OSI model has other uses for the word "packet", we'll refer to an OSI frame here also as either "packet" or "message". Higher levels in the OSI model don't really apply well to this topic and we leave them out of consideration here.

OS Version 2.1 and 3.0 Protocol Data Frames

Message data is best described in a “nibble-oriented” fashion (a nibble is 4-bits). Figure 1 depicts the message structure of version 2.1 and 3.0 messages. The size of each block (in nibbles) is given in parentheses.

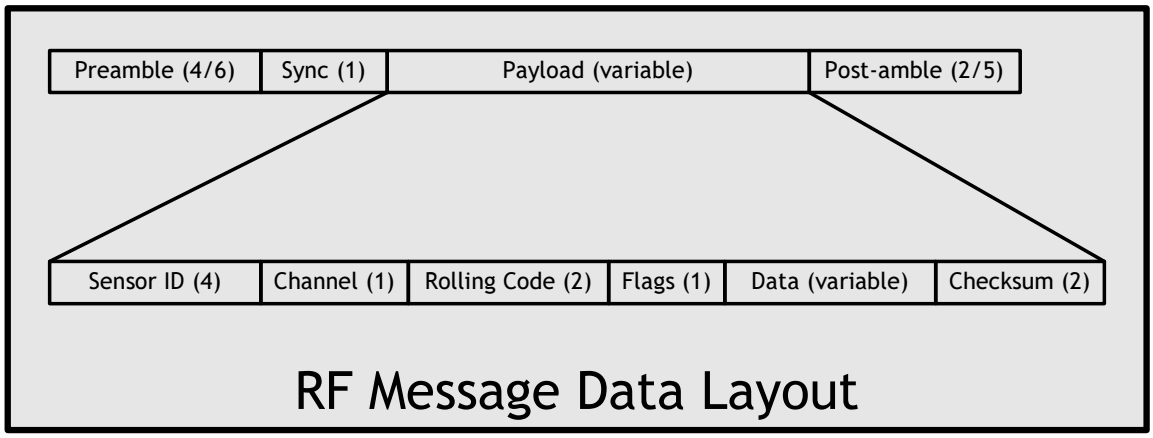

Figure 1. Layout of version 2.1 and 3.0 messages

Both 2.1 and 3.0 protocols have a similar message structure containing four parts.

1. The preamble consists of “1” bits, 24 bits (6 nibbles) for v3.0 sensors and 16 bits (4 nibbles) for v2.1 sensors (since a v2.1 sensor bit stream contains an inverted and interleaved copy of the data bits, there is in fact a 32 bit sequence of alternating “0” and “1” bits in the preamble).

2. A sync nibble (4-bits) which is “0101” in the order of transmission. With v2.1 sensors this actually appears as “10011001”. Since nibbles are sent LSB first, the preamble nibble is actually “1010” or a hexadecimal “A”.

3. The sensor data payload, which is described in the “Message Formats” section below.

4. A post-amble (usually) containing two nibbles, they seem to be a CRC-8 checksum but the specifics vary from sensor to sensor. At least one sensor (THR238NF) sends a 5-nibble post-amble where the last four nibbles are all zero.

The number of bits in each message is sensor-dependent. The duration of most v3.0 messages is about 100msec. Since each bit is doubled in v2.1 messages, and each v2.1 message is repeated once in its entirety, these messages last about four times as long, or 400msec.

OS Version 1.0 Data Frames

Version 1.0 sensors have a simpler format as shown in figure 2 below.

1. The preamble contains twelve “1” bits.
2. The sync section consists of a long off period (4.2msec), a long RF pulse (5.7msec) and another long off period (around 5msec).
3. The first data sample point (clock edge) is not always marked by an RF transition and must be measured from the end of the long sync pulse.
4. The data payload is fixed length since all version 1.0 sensors can only measure temperature.
5. These sensors do not transmit a post-amble.

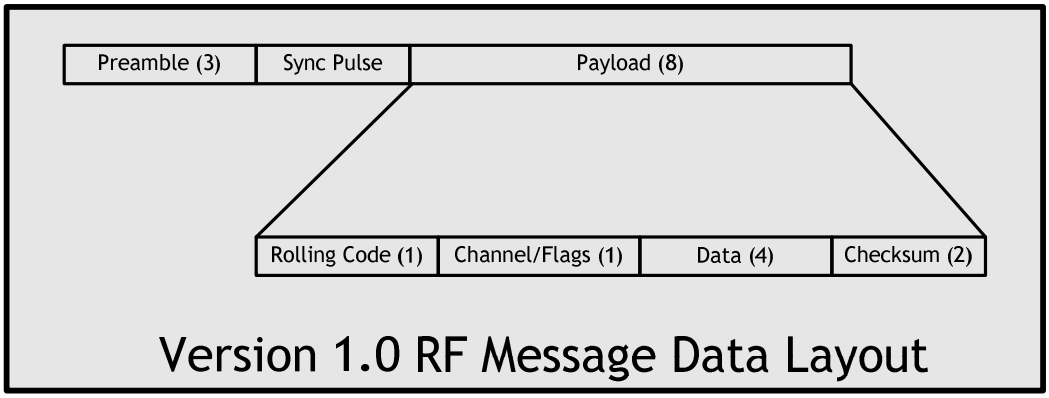

Figure 2. Layout of version 1.0 messages

AcuRite VN1TX (aka 5-in-1) Data Frames

Figure 3 shows this layout. It is a bit more complex than other formats in the sense that information is substantially split across byte and nibble boundaries.

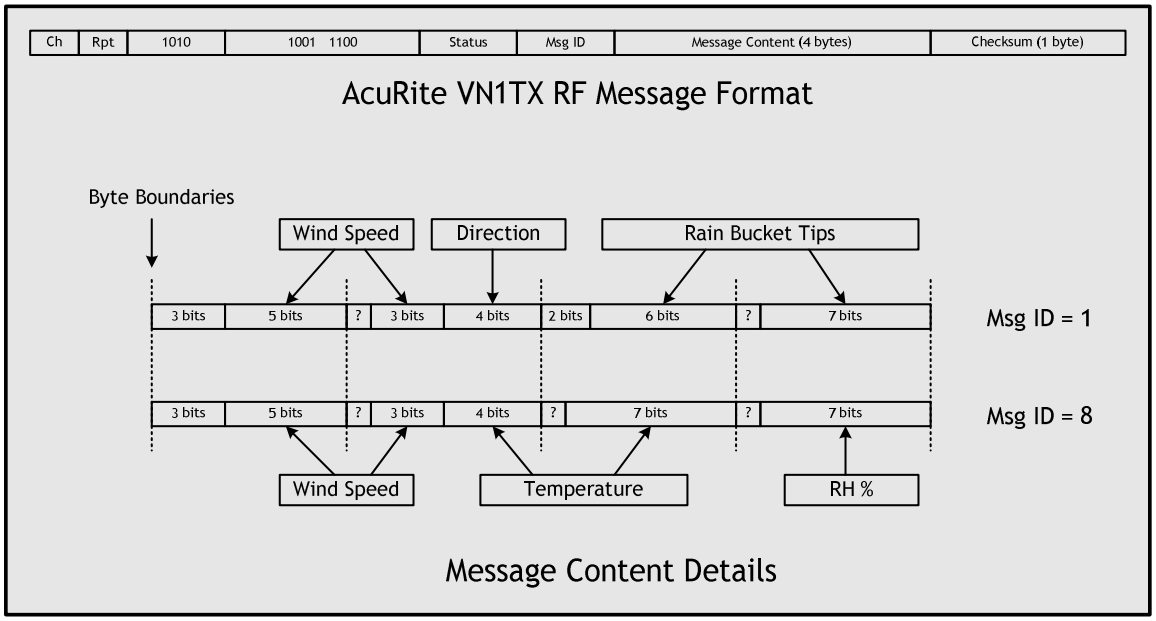

Figure 3. Acu-Rite VN1TX Message Format

The checksums are simple modulo-256 sums of all preceding bytes (not a sum of nibbles).

There are two types of message sent; one contains wind speed/direction and rainfall data. The other provides wind speed, temperature and humidity values.

One post on the *wxforum* web site[1] stated that reported wind speed is the highest speed measured during each 18-second period within a 4-second sampling window. See below for conversion of the reported integer value to km/hr. Wind direction mapping from integer value to direction is also convoluted.

Temperature is unsigned in units of 0.1 degF above -40F. Divide the value by ten and subtract 40 to get temperature in degrees F.

Rain is just bucket tip count, with 0.01 inch per bucket tip.

The fifth nibble (upper nibble of the 3rd byte) contains status information. Right now it is only known that with all okay, the binary value is '0111'. This

[1] http://www.wxforum.net/index.php?topic=25705.msg247980

changes to '1011' when batteries are low. It is possible that bit 3 is a flag bit, summarizing all status while bit 2 is a "battery okay" bit. These are just guesses at this point.

When batteries get very low, one unit tested stops sending the type "1" message, and the third repetition of data in the type "8" message contained different temperature/humidity values than the first two repetitions.

There are currently several unknown bits in this message format. For example, nibbles 2,3,4 always seem to be "A9C"; is this an identifier sequence or something else?

When batteries get really low, the unit seems to cease sending the wind/rain message and the data in the third message repetition has a different value for temperature and humidity than the first two copies. The actual values are a temperature of 49.4F and a humidity of 95%.

VN1TX Wind Speed Conversion

Raw sensor data is reported for wind speed: the number of full cup rotations measured in a 4-second period. Furthermore, being reported once every 18 seconds, the value is the highest 4-second rotation count during the 18-second period.

The VIS reader app for AcuRite shows wind speed in kph to two decimals. Comparing the data record numbers to reported speed by VIS reader in km/hr yields the following wind speed formula.

$$km/hr = spd \times 0.8278 + 1.00$$

where spd is the integer value from the message (revolutions per four seconds). The addition of the constant is performed unless spd is zero in which case kph is also zero. This equation comes from a least-squares fit to VIS reader values and fits their reported wind speeds to within +-0.005kph (e.g. one-half significant digit).

Expressing this in cm/sec and converting "spd" to revolutions per second (f = spd/4), we have:

$$cm/sec = f \times 91.97 + 27.78$$

Physical measurements of the anemometer yield a cup center radius of Rrc=5.2cm and a cup half-width of Rc=2.2cm. From this we can calculate the anemometer's K-factor:

$K = 91.97 / (2 \times PI \times 5.2) = 2.8$

This is fairly low as K-factors go but is in the realm of a believable number. And for those interested in anemometer design, the Rc/Rrc ratio is about 0.42 for this anemometer.

AcuRite 00592TXR Tower Data Frames

Packets are seven bytes long with a single extraneous zero bit at the end which can be discarded. Data is transmitted in big-endian order per byte, and multi-byte data is also in big-endian order.

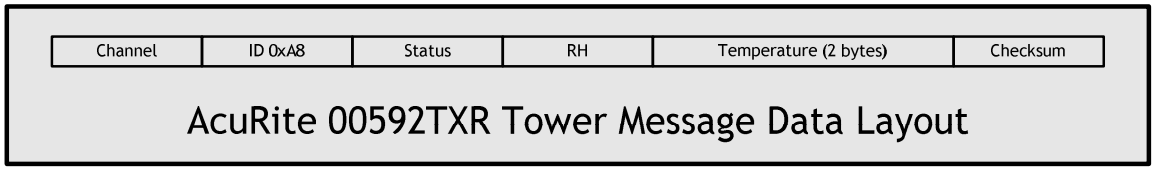

Figure 4. AcuRite 00592TXR Tower Message Data Format

The channel is encoded in the two MS bits of the byte, A=11, B=10, C=00. The remaining bits of the channel byte seem to be static and are 100000. The status byte is 0x44 when battery voltage is above 2.5 volts and 0x84 when battery is below 2.5 volts.

The RH and temperature bytes only contain seven bits of data each, the MSB is a parity bit for even parity and this can be used as part of a frame validity check. None of the other message bytes appear to contain parity bits. As a result, RH data consists of seven bits and temperature data contains 14 bits.

RH is in simple binary with units of percent.

Temperature is in 0.1 degree Celsius units with a 100.0C offset. To get temperature then, divide the binary value by 10 and subtract 100. To get the binary value, shift the seven LSBs of the first temperature byte left 7 bits and add the 7 LSBs of the second temperature byte.

The checksum is a simple modulo-256 sum of the preceding six bytes.

PSM Data Frames

Figure 5 shows these two message layouts. The SL-109H includes a 4-bit checksum while the AcuRite 00955 message has none. Both of these message formats transmit each nibble in MSB-first order -- opposite to the order used by all other OS formats.

SL-109H

The SL-109H message is 38-bits long and begins with a 4-bit checksum. This checksum is computed by right-aligning the two-bit channel number in a nibble, then summing in with the rest of the message nibbles. Relative

humidity is sent in percent, in BCD format with the MSD first, while temperature is a 12-bit signed binary value with the LSB equal to 0.1 degrees Celsius. Currently, the contents of the status nibble are unknown -- or they may serve some other purpose.

Here is an example of an SL-109H message:

6-0-560C142C

The first nibble (6) is followed by a 2-bit channel code ("-0-"), then the rest of the message nibbles. We have the following values for this message:

- Checksum nibble is 6
- Channel code is 0 (two bits)
- Relative humidity is 56%
- Temperature in deg C is (0.1 x 0x0C1) = 0.1 x 193 = 19.3 deg C
- Status nibble is zero
- Rolling code is 0x2C

For each transmission, this message is typically repeated three times.

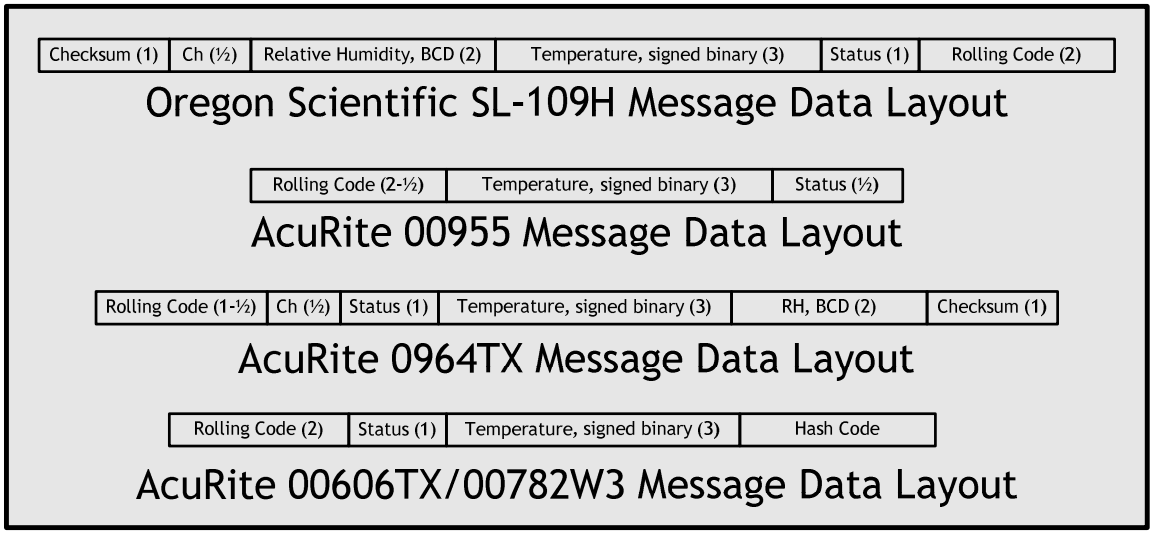

Figure 5. AcuRite and SL-109H Formats

AcuRite 00955

A typical 24-bit message from this sensor would look like this:

1270C18

Here, the first two bits have been placed into the first nibble's LSBs and the last nibble contains only two message bits in its MSBs. As indicated in figure 3, we have:

- Rolling code is 0x127
- Temperature in deg C is (0.1 x 0x0C1) = 0.1 x 193 = 19.3 deg C
- Status bits are (in binary) "10" (these are the two MSBs from the last nibble). One of these bits seem to indicate whether the transmission was caused by pressing the TX button or not. The other bit may be battery status, but that needs to be verified.

AcuRite 0964TX Messages

This sensor transmits PSM just like the 00955 model, but nibbles are sent LSB first (unlike the 00955), and are 9 nibbles long (36 bits). For example, the value "3" is sent as the series of bits "1100" in little endian order -- not "0011" which would be big endian order.

Channel number switch setting is encoded in the two LSBs of the 2nd nibble. The correspondence between channel switch and the bits is as follows:

```
Ch 1 ==> 0x02, Ch 2 ==> 0x01, Ch 3 ==> 0x11
```

Multiple-nibble values are sent in little-endian order. An example 0964TX message is below (these nibbles have been interpreted in little endian order):

8602EF020

The "8" and two MSBs of the 2nd nibble are a rolling code; for this example it is '1000 01' in binary. This would be represented as '01 1000' taking the second nibble as the big endian so the rolling code could be called 0x21 in big endian order or 0x18 in little endian order. The rolling code seems to vary with temperature, humidity and battery voltage.

The next two bits ('10') are the channel number, in this case the switch is set to position "1".

This is followed by the status nibble of zero; the LSB will become a one if the battery is low. This transition occurs somewhere around 2.6 volts.

Next is the temperature "2EF" is in little endian order, so is really "FE2" as a signed value and is equal to -30 which represents -3.0C.

The BCD RH value is 20%, but in little endian order is "02". Finally the check nibble is the sum of all previous nibbles modulo 16 but with all bits inverted. (The sum modulo-16 here is 0xF, which inverts to 0x0).

AcuRite 00606TX and 00782W3

These sensors have identical formats and add yet more variation in data encoding. Data bits are sent in big-endian order and nibbles are also in big-endian order (opposite to the 0964TX sensor). The data packet length is also different at 32 bits.

The status nibble's MSB is a "battery okay" flag which is normally one. It goes to zero when the battery drops below about 2.6 volts or so.

Temperature is a signed 12-bit value with resolution of 0.1C. For example, 25.6C is encoded as the value 256 (0x100 hexadecimal). A value of -0.1C is encoded as -1 (0xFFF hexadecimal).

There is an 8-bit hash code appended to check message integrity. It is the same algorithm as described for the Ambient Weather F007TH sensor below with two minor modifications.

1. The hash code value is initialized to zero instead of hexadecimal "0x64".

2. Start with the fifth value in the LFSR sequence depicted for the F007TH sensor -- 0xF1 instead of the first value, 0x3E.

Ambient Weather WH2C Sensor

The full message contains six bytes or 12 nibbles. The preamble is one byte and consists of 8 ones.

This sensor provides an 8-bit CRC computed on all message data except the preamble, using a polynomial of 0x131. The register initialization is zero for this CRC. Nibbles are sent MSB first, and multiple-nibble fields are in big-endian order.

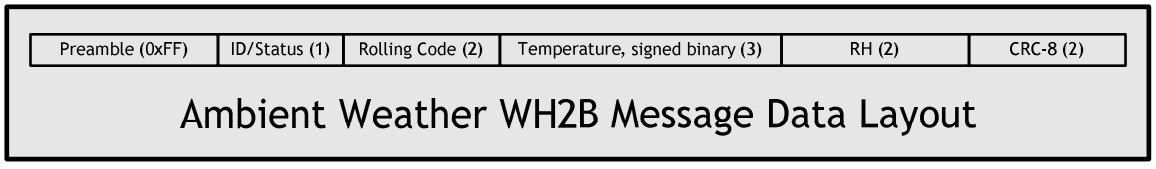

Figure 6. Ambient Weather WH2B Data Format

Below is a sample WH2C message:

4950FA3D4E

The rolling code here is "95", temperature is "0FA" (250 decimal) which represents 25.0C and the humidity is "3D" or 61%. The CRC-8 byte is "4E" which includes all data starting with the ID/status nibble.

A low-battery condition does not appear to be part of the message and the unit will transmit messages with battery voltages as low as 1.7 volts (fresh batteries are 3.0 volts).

Ambient Weather F007TH

Each data frame here is six bytes, with the entire message including preamble is sent three times with no delay between repetitions. All data is sent big-endian order, bits and bytes.

Because the preamble is 13 bits, each successive message repetition is shifted one bit relative to byte or nibble boundaries. Extracting the repeated messages therefore requires a one or two-bit shift for the 2nd and 3rd copies respectively.

The fixed ID is followed by a one byte rolling code. The channel number is in the lower three bits of the next nibble. The upper bit always seems to be zero and no low battery indication is apparent anywhere in the data frame.

Temperature is encoded in the next 12 bits. It is in units of tenths of a degree Fahrenheit with a -40 degree offset. Multiply the integer value by 0.1 and subtract 40 to get temperature in Fahrenheit.

Relative humidity is in the next byte as an integer value in percent.

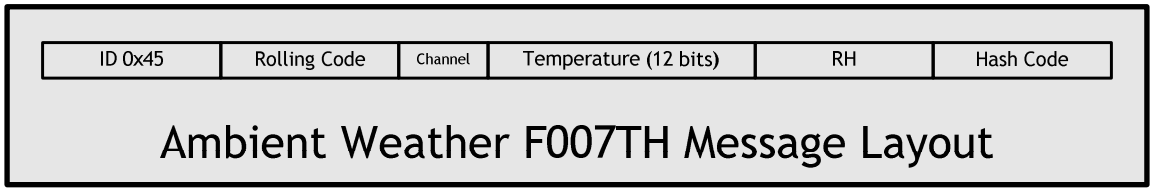

Figure 7. Ambient Weather F007TH Data Format

F007TH Hash Code

This sensor does not use a simple checksum or even a more advanced CRC. Instead data integrity is verified by a hash code, generated by multiplying the message bits with the byte sequence generated by a linear feedback shift register (LFSR). This algorithm has been referred to as "an LFSR-based Toeplitz hash" in some of the literature. We can thank "Ron" at this blog:

https://eclecticmusingsofachaoticmind.wordpress.com/2015/01/21/home-automation-temperature-sensors/

for reverse-engineering this hash algorithm. Great job Ron!

The easiest way to explain the algorithm is in two parts. The first part is the LSFR design used to generate a sequence of bytes.

Start with an 8-bit register initialized to the value 0x7C. Generate a number of values equal to the message size in bits by repeating the following operations once for each bit.

- Rotate the register right one bit.
- If the bit shifted out of the LSB and into the MSB during the rotation was a one, then exclusive-or the value 0x18 into the register (i.e. flip the state of bits 3 and 4). Do this ***after*** the rotate operation.
- The register value ***after*** these two operations is the sequence value to be stored.

Perform these steps once for each bit in the message. If the message contains 40 bits for example (as with the F007TH message) then you will need to generate a sequence of 40 bytes. Because this sequence does not depend on the data message contents and can be pre-computed once and stored to save time.

The second part of the hash algorithm combines the LSFR sequence with message bits to form the final hash value.

To compute the message hash value, sequence through the 40 message bits in the order they were received, starting with the ID byte (0x45). Since everything here is big-endian, that means proceeding from MSB to LSB.

Start by initializing the hash register to the value 0x64. As the message bits are read, for every bit in the message that is a one, exclusive-or the corresponding value from the LSFR sequence into the hash register. For example, if the 17th bit is a one, then take the 17th value from the LFSR sequence and exclusive-or it into the hash register. If the message were all zeros, then nothing would be added to the hash register and the result would be the initial value of 0x64.

LSFR Sequence Details

For clarity, the LFSR design used by the F007TH is shown pictorially below. The initial bit values (0x7C) are show in the boxes.

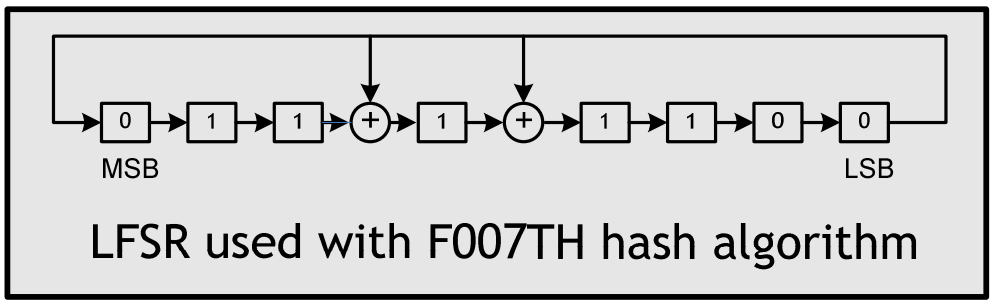

Figure 8. Linear Feedback Shift Register

This particular LFSR sequence repeats every 127 values. Therefore, even for a very long message, it would only necessary to compute a table of 127 values and use the message bit number modulo 127 as an index into the table.

For reference, below are the 127 hexadecimal values generated by this LFSR. Only the first 40 values are required for use with F007TH data frames.

```
3e 1f 97 d3 f1 e0 70 38 1c 0e 07 9b d5 f2 79 a4
52 29 8c 46 23 89 dc 6e 37 83 d9 f4 7a 3d 86 43
b9 c4 62 31 80 40 20 10 08 04 02 01 98 4c 26 13
91 d0 68 34 1a 0d 9e 4f bf c7 fb e5 ea 75 a2 51
b0 58 2c 16 0b 9d d6 6b ad ce 67 ab cd fe 7f a7
cb fd e6 73 a1 c8 64 32 19 94 4a 25 8a 45 ba 5d
b6 5b b5 c2 61 a8 54 2a 15 92 49 bc 5e 2f 8f df
f7 e3 e9 ec 76 3b 85 da 6d ae 57 b3 c1 f8 7c ..
```

Rolling Codes

Many sensors use rolling codes as a means to reduce the likelihood of interference between neighbors. These are typically pseudo-random values with the intention that a sensor will power up with a different rolling code every time batteries are installed. With some sensors, these are often just based on current measurement values of temperature and/or humidity; if a sensor is powered up with the exact same temperature and humidity then the rolling code may always be the same. Other sensors may introduce additional random factors to prevent this.

Either way, rolling codes are both a blessing and a curse. The effectively increase the number of discrete channel settings for each type of sensor. On the other hand, because rolling codes cannot be set directly by the user they are also a nuisance.

Protocol Summary

The table below summarizes the differences between the three different versions.

Protocol Version	Bit Rate (Hz)	Manchester Polarity	Preamble Bit Count	Bits Doubled	Message Repeated	RF Pulse Length Offset
1.0	342	Reverse	12	No	Yes	+255 μsec
2.1	1024	Normal	32	Yes	Yes	-96 μsec
3.0	1024	Normal	24	No	No	-138 μsec

Decoding RF Messages

This section describes techniques which may be used to decode data frames from the various sensors described above.

Decoding Hardware

Reception and decoding is possible using an Arduino board combined with one of the inexpensive 434MHz receiver modules which are readily available.

Hardware available for decoding on the Atmel processor includes a hardware timer with an edge-triggered sampling input. Edges on the trigger input will cause the timer value to be latched and a processor interrupt is then generated. One easy way to handle decoding then is not to attempt clock recovery, but to examine the time difference between transitions on the OOK RF signal.

Classifying Time Intervals

The decoding algorithm works by classifying time intervals between RF transitions (on-to-off and off-to-on) as either short or long.

Because the RF pulses are shortened, separate time thresholds are used for classifying the time period (short or long) depending on the RF state (on or off). Based on the data rate (1024Hz) and the two amounts by which pulses are shortened (96us and 138us), the table below shows the expected values of time intervals (in microseconds) based on the protocol version and RF state.

There is a wider range of timing variability with version 1.0 sensors and the table indicates the range of values seen among different sensors.

Protocol Version	RF On		RF Off	
	Short	Long	Short	Long
Version 2.1	396	884	581	1069
Version 3.0	349	837	628	1116
Version 1.0 (preamble/data)	1450-1720	2920-3180	1219-1480	2680-2940
Version 1.0 leading sync off	--	--	--	4200-4500
Version 1.0 sync pulse	--	5500-5700	--	--
Version 1.0 trailing sync off	--	--	--	5200-5500
AcuRite RF pulse width	400	600		
AcuRite short off period			1700	2400
AcuRite long off period			2400	4300
AcuRite separator off period			8500	10000

AcuRite VN1TX PWM	198-228	385-422		
AcuRite VN1TX sync	605-678		605-678	

With the VN1TX, RF off interval length is not of any concern when data is being sent because it is only the RF on pulse width which determines the data bit. When looking at the AcuRite pulse widths at the input signal to the transmitter, values are very stable, 214usec for short pulses (except last interval in each group which is 207usec, 405-406usec for long pulses and 612-613usec for sync. The variation seen above is assumed to be mostly due to the receiver, although there could be some variation in the time between the input signal to the transmitter and the actual start of the RF pulse.

Averaged thresholds for classifying time intervals as short or long have been determined. Times given in the table below are in microseconds. Time intervals which fall outside the “Short Min” or “Long Max” values are considered invalid. These are for version 2.1 and 3.0 sensors.

RF State	Short Min	Short Max	Long Min	Long Max
Off	400	850	850	1400
On	200	615	615	1100

These averaged thresholds only vary by about 20 μsec from the ideal threshold that would be chosen for either version of sensor (2.1 versus 3.0).

A small improvement in performance might be possible by using different threshold values for each protocol version – which would be possible after the preamble is identified and the protocol version is known.

Additional measurements of version 1 sensors have show a large variability in the transmitted time intervals. RF "on" pulses as short at 1450 usec have been seen, suggesting that the lower limit should probably be set just above the maximum on pulse length for version 2.1 and 3.0 sensors (which is 1400usec). The current set of values in the table below are used in the current Arduino sketch for WSDL WxShield.

A reasonable set of thresholds for version 1.0 sensors (in micro-seconds) is shown in the table below:

RF State	Short Min	Short Max	Long Min	Long Max
Off	970	1950	1950	3100
On	1404	2400	2400	3400
Sync Begin (Off)			4000	4600
Sync (On)			5400	5985

Sync End (Off)			5000	5600
Sync End (Off)			6480	7100

Note: The two “Sync End” intervals correspond to the two cases where the first data bit is a “1” or “0” respectively.

Decoding Using Time Intervals

The decoding algorithm works by capturing a timer value when RF transitions (on-to-off or off-to-on) occur, and calculating the time interval between successive transitions. These intervals are classified as either short (one-half clock period) or long (one full clock period).

An integer counter keeps track of time in units of one-half clock tick; this counter’s value will be called “half-time”. After being properly initialized, half-time is incremented by one when a short interval occurs and by two for long intervals. Half time is a very useful quantity for decoding RF messages:

- When half-time is even, we are at the middle of a clock period. The transition occurring at this point determines the bit being transmitted.
- When half-time is odd we are at the boundary between two clock periods. Transitions occurring here are of no interest in determining transmitted bits.
- When half time is even, dividing it by two yields the current message bit number[2].

Using half-time, some very simple logic can be used to decode the RF signal.

Decoding Messages

When a transition falls on a boundary between two clock periods (i.e. half-time is odd), there is no message bit to be decoded. There may still be some useful information here however; if the current time period is long it means that the last transition also occurred at a clock period boundary. This means that there was no transition in the middle of the currently ending clock period, and signifies a violation of Manchester-coding format. This should be detected as an error condition.

[2] For version 1.0 and 3.0 sensors only -- for version 2.1 sensors, half-time must be divided by four to get the message bit number.

When a transition falls in the middle of a clock period (half-time is even), a message bit can be detected and its value is simply equal to the RF state (on=”1” and off=”0”) just prior to the transition.

The decoding algorithm described above is simple and correctly determines the polarity of each bit based on the current RF signal state (on/off). Another algorithm has been developed by others which also works but does not consider the RF state when detecting bits (except for the first bit). This algorithm is described later.

The half-time value is also useful for verifying that bit-doubling is correct in version 2.1 messages. Since a long transition period is required to change from a 1 to a 0 bit (or vice-versa), every bit pair in these messages is required to end with a long transition period. Furthermore, when time is aligned with the end of a double-bit period, half-time taken modulo-4 will be zero.

When decoding a message from a version 2.1 sensor, and half-time modulo-4 is non-zero, no bit is detected. When half-time modulo-4 is zero, a bit is detected and a check is made that the current transition period is a long one (otherwise an error exists).

Initializing Half-Time

As mentioned above, half-time must be initialized correctly for the algorithm to work. This translates to two requirements:

1. Half-time must be initialized to an even value at a transition that occurs in the middle of a clock period.

2. If half-time will be used to track message bit numbers, half time modulo-2 (or modulo-4 for version 2.1 sensors) must be set such that it represents the correct bit number (possibly with an offset) at an early point in the message. The logical place for this initialization is when the end of the preamble is detected.

AcuRite VN1TX Challenges

Decoding this sensor is simple as long as OS3 protocol messages are not also present. When both VN1TX and OS3 messages are present, care must be taken in properly delineating the OS3 preamble from the VN1TX preamble.

Confusion can occur because the transition time interval on the sync pulses are in the same range as times for the first long time interval after the OS3 preamble. The first RF pulse after preamble is a bit shorter and is followed by a short off interval and these two events can be used to identify this sensor's signal.

Alternate Algorithms

These algorithms have been published on the internet previously by various people.

As will be shown below, if the value of the first message bit is known then the message can be decoded by considering only the time intervals between RF state transitions, and ignoring the actual RF state value at each transition.

In a Manchester coded signal, each source data bit generates either a pair of short transition intervals or a single long transition interval. A source bit will generate a pair of short transition intervals when it is the same value as the preceding source bit. When a source bit has the opposite value as the preceding source bit, a single long transition interval is generated.

This description of Manchester coding lends itself to decoding based solely on transition timing. A pair of short transitions represents a bit identical to the previous bit. A long transition means the current bit is the opposite of the previous bit. This works as long as the value of the first bit can be correctly determined – otherwise the resulting decoded bit stream will be inverted.

Here is another algorithm that will properly decode version 2.1 messages: every long period represents no change in bit state while every pair of short periods represents the bit state changing. Under this interpretation, the preamble decodes as 32 “1” bits instead of a repeating “1010…” pattern. Furthermore, each bit in the message appears doubled without inversion – the sync nibble would be “00110011” for example. Answering the question of why this works is an exercise left for the reader.

Decoding AcuRite PSM Messages

These message are conceptually much easier to decode. Each "on" pulse should be checked for a proper duration, and the length of the "off" periods is used to determine each bit (one or zero).

The presence of these messages can be identified by the appearance of a 2 msec long off period. This long of an off period does not occur when noise is being decoded and luckily, the AGC circuits in the MC33596 receiver used in the WxShield will not ramp the gains back up to noise detection levels in only 2 msec (although they will in a 4 msec off period). This permits the detection of an AcuRite message -- but the determination is often made after the first bit has been received and one or more bits may have been lost.

This problem is easy to solve as there are at least three repetitions of each message. After the AcuRite message is suspected, receiver AGC can be gated so as to only be enabled when the RF signal is on and in this way, over the length of the first message repetition will converge on the proper gain setting. When

the first long off period (9 msec) between repetitions is detected, a transition to recording message bits is made and the AGC can be periodically turned on once or more during each message if desired.

In practice, this additional decoding logic does not seem to interfere with detection and decoding of version 1.0, 2.1 and 3.0 RF signals.

There may be other wireless sensors that transmit in this format but this is not covered here.

When an AcuRite PSM format message is received, it must be carefully examined to determine whether it represents an AcuRite temperature-only message or an Oregon Scientific SL-109H temperature/humidity message. The presence of additional sensors using this RF protocol will further complicate this task.

One possible approach is to look for the number of bits over which the message repeats. AcuRite 00955 messages are 24 bits in length, while SL-109H messages are 38 bits long.

Message Formats

All OS version 2.1 and 3.0 messages decoded so far appear to have an identical format for the sensor data payload, as shown in the table below. Figure 1 (earlier in this document) depicts the payload format. The message is assumed to contain “n” nibbles, numbered from zero to n-1. For convenience, this table also shows the checksum and post-amble portions of the message.

The coding of sensor-specific data varies according to the type of measurements being reported by the sensor. Some sensors use the same coding as others which report the similar data – but this is not always the case. For example, the THGR810 and THGR122NX temperature/humidity sensors use the same data coding, but the RGR968 and PCR800 rain gauges do not.

Nibble(s)	Contents	Details
0..3	Sensor ID	This 16-bit value is unique to each sensor, or sometimes a group of sensors.
4	Channel	Some sensors use the coding 1 << (ch – 1), where ch is 1, 2 or 3.
5..6	Rolling Code	Value changes randomly every time the sensor is reset
7	Flags 1	Bit value 0x4 is the battery low flag
8..[n-5]	Sensor-specific Data	Usually in BCD format
[n-3]..[n-4]	Checksum	The 8-bit sum of nibbles 0..[n-5]
[n-1]..[n-2]	Post-amble	Unknown contents and purpose

Most (but not all) sensor data is in BCD format, least significant digit (LSD) first. For example the value of 27.5 degrees Celsius would be coded as the hex nibbles “572”. The decimal point is assumed to be in a fixed location and is not actually transmitted.

Version 1.0 Message Format

At this point, there is only a single known format for version 1.0 messages. All version 1.0 sensors are temperature-only units.

Nibble(s)	Contents	Details
0	Rolling Code	This 8-bit value changes randomly when the sensor is reset or batteries changed.
1	Channel	Channels 1,2,3 are coded as 0,4,8
5..2	Temperature	BCD temperature in degrees Centigrade
7..6	Checksum	Byte-oriented checksum

Version 1.0 messages are 8 nibbles in length. The channel setting occupies only two bits in nibble 1 and it is possible that the other two bits may be part of the rolling code. They have occasionally been seen to be non-zero.

The rolling code does not change every time the reset button is pressed. Several reset operations are usually required to get this code to change.

According to internet sources, the first temperature nibble (nibble 5) is actually a bit status field containing the following bits:

- 0 – Not used
- 1 – A “1” value indicates negative temperature
- 2 – Unknown (may be a malfunction flag)
- 3 – Battery low when “1”

The checksum is computed by organizing the 8 nibbles into four bytes in little endian order. Any overflow is summed back into the total sum.

For example, a message received as (in the order of transmission) “8487101C” would contain the following bytes: 0x48, 0x78, 0x01, 0xC1. The first three bytes are summed and compared to the checksum (0xC1 in this example). This message contains a rolling code of “8” and the sensor is set to channel 2, reading 17.8 °C.

For another example take the message “88190AAB”. The bytes 0x88, 0x91 and 0xA0 sum to a value of 0x1B9. The overflow (0x1) is summed back in giving a final checksum of 0xBA. This sensor has a rolling code of “8”, is set to channel 3, reads -9.1 ºC and has a low battery.

Notes on the Status nibble

- There have been reports that one or more bits are reporting a "comfort" or similar value. This is unconfirmed by this document's author, but may be true.

- One of the status bits may flip between the two repetitions of version 2.1 RF messages -- with unknown meaning. Again, unconfirmed.

Known Sensor ID Codes

These are the currently known codes for both version 2.1 and version 3 sensors.

Sensor	Code	Sensor	Code	Sensor	Code
BTHR918	5A5D	BTHR968	5D60	PCR800	2914
PSR01		RGR918	2A1D	RGR968	2D10
RTGR328NA				STR918	3A0D
THC268		THGN123N	1D20	THGN801	F824
THGR122NX	1D20	THGR228N	1A2D	THGR268	
THGR810	F824	THGR810[1]	F8B4	THGR918	1A3D
THN132N	EC40	THR238NF	EC40	THR268	
THWR288A	EA4C	THWR288A-JD		THWR800	C844
UVN800	D874	UVR128	EC70	WGR800[2]	1994
WGR800[3]	1984	WGR918	3A0D	THGN500	1D30
BTHGN129	5D53				

Footnotes:

1. This is the temperature/RH sensor that originally shipped with the WMR100 – it was integrated with the anemometer.

2. The original anemometer which included a temperature/RH sensor.

3. The newer anemometer with no temperature/RH sensor.

Nibble values in these codes assume LSB first order. That is, if the bits of a nibble in order of transmission are ‘0101’, the hex value is taken to be ‘A’ (not ‘5’).

The nibbles are presented in order of transmission. However, since all other multi-nibble data in the sensor data message is sent least-significant nibble first these values might be considered “backwards”. In other words, the ID code “1D20” shown above might be more properly called “02D1”. That said, this description describes the code nibbles in order of transmission.

Known Sensor Data Formats

These tables number the message nibbles starting with the sensor ID, so the first nibble of sensor data is contained in nibble 8. Message lengths include the checksum, but not the two final nibbles (for which the content is unknown).

ID Code(s)		Message Length (nibbles)
1D20, F824, F8B4		17
Nibbles	Contents	Temperature/Humidity
10..8	Temperature	LSD is 0.1 degC
11	Temperature Sign	Non-zero for negative values
13..12	Relative Humidity	Percent
14	Unknown	

ID Code(s)		Message Length (nibbles)
EC40, C844(?)		14
Nibbles	Contents	Temperature Only
10..8	Temperature	LSD is 0.1 degC
11	Temperature Sign	Non-zero for negative values

ID Code(s)		Message Length (nibbles)
EC70		14
Nibbles	Contents	Ultra-violet
9..8	UV Index	Unit-less Integer
11..10	Unknown	

ID Code(s)		Message Length (nibbles)
D874		15
Nibbles	Contents	Ultra-violet
12..11	UV Index	Unit-less Integer
10..8	Unknown	

ID Code(s)		Message Length (nibbles)
1984, 1994		19
Nibbles	Contents	Anemometer
8	Direction	Not BCD – binary value from 0..15. Direction in degrees is value * 22.5 degrees.
9	Unknown	
10	Unknown	
13..11	Current Speed	In meters per second, LSD is 0.1m/s
16..14	Average Speed	Same as above

ID Code(s)		Message Length (nibbles)
2914		20
Nibbles	Contents	Rain Gauge
11..8	Rain Rate	LSD is 0.01 inches per hour
17..12	Total Rain	LSD is 0.001 inch

ID Code(s)		Message Length (nibbles)
2D10		18 (?)
Nibbles	Contents	Rain Gauge
10..8	Rain Rate	LSD is 0.1 mm per hour
15..11	Total Rain	LSD is 0.1 mm

ID Code(s)		Message Length (nibbles)
5D60		21
Nibbles	Contents	Temp/RH plus Barometer
10..8	Temperature	LSD is 0.1 degC
11	Temperature Sign	Non-zero for negative values
13..12	Relative Humidity	Percent
14	Comfort Level	0: normal, 4: comfortable, 8: dry, C: wet
15	Unknown	
18..16	Pressure	Binary (not BCD) in units of 0.01 inHg

ID Code(s)		Message Length (nibbles)
5D35		21
Nibbles	Contents	Temp/RH plus Barometer
10..8	Temperature	LSD is 0.1 degC
11	Temperature Sign	Non-zero for negative values
13..12	Relative Humidity	Percent
16..14	Pressure	Binary (not BCD) in units of 0.01 inHg
17..18	Unknown	

Examples

Below are some examples of properly decoded transmissions from a THGR122NX sensor. Messages are listed as a string of hexadecimal nibble values, in the time-order they were received. This sensor represents the channel switch setting as the value (1 << (channel-1)), so channels 1,2,3 appear as the values 1, 2, 4.

1D20485C480882835

This sensor is set to channel 3 (1 << (3-1)) and has a rolling ID code of 0x85. The first flag nibble (0xC) contains the battery low flag bit (0x4). The temperature is -8.4 ºC since nibbles 11..8 are “8084”. The first “8” indicates a negative temperature and the next three (“084”) represent the decimal value 8.4. Humidity is 28% and the checksum byte is 0x53 and is valid.

1D2016B1091073A14

This sensor is set to channel 1 (1 << (1-1)) and has a rolling ID code of 0x6B, and the battery low bit is not set in the flag nibble (0x1). Temperature and humidity are 19.0 ºC and 37%. Checksum is 0x41 and is valid.

Detecting Bad Data

These RF protocols use a simple arithmetic checksum to provide data integrity. This does not in fact provide adequate protection against data corruption. From time to time, corrupted messages with valid checksums will be received. The likelihood of this increases as more wireless sensors are added to a weather station. Additional validity checks can often identify these bad apples:

- For version 2.1 protocol messages, instead of just blindly discarding every other bit, verify that each bit pair is either ‘10’ or ‘01’.
- Verify that the record length is correct for the sensor ID code.
- If the record represents a new sensor (according to the combination of ID code, rolling code and channel), wait until it is received at least twice within a 2-3 minute period before assuming it is truly a new sensor.
- Test all nibbles that should contain BCD digits (i.e. hex values A through F are not valid).
- Validate any other nibbles that have a limited set of valid settings.
- Perform sanity checks on the decoded numbers.

Appendix I

Sample Recordings

Several recordings captured on an oscilloscope are shown here to help in visualizing the RF protocols described previously in this document. The horizontal time axis has been calibrated in clock periods. This makes it much easier to visualize the data being represented by on and off periods of the RF signal. The vertical axis is unit-less and simply indicates whether the RF signal is on (the higher level) or off (the lower level). Integer values on the horizontal axis are aligned with the middle of each clock period – not the boundary between clock periods.

Version 3.0 Protocol Samples

This first example shows a version 3.0 preamble sequence. Consisting of all "1" bits, the Manchester coding requires that the RF signal be "on" immediately prior to each clock transition. To achieve this, the RF signal must be turned off at the start of the clock period so that it can be turned back on prior to the end of the clock period. Remember, that Manchester coding requires there to be an RF transition (on-to-off or off-to-on) at the end of each clock period, so it is not possible to simply leave the RF signal on during the entire preamble.

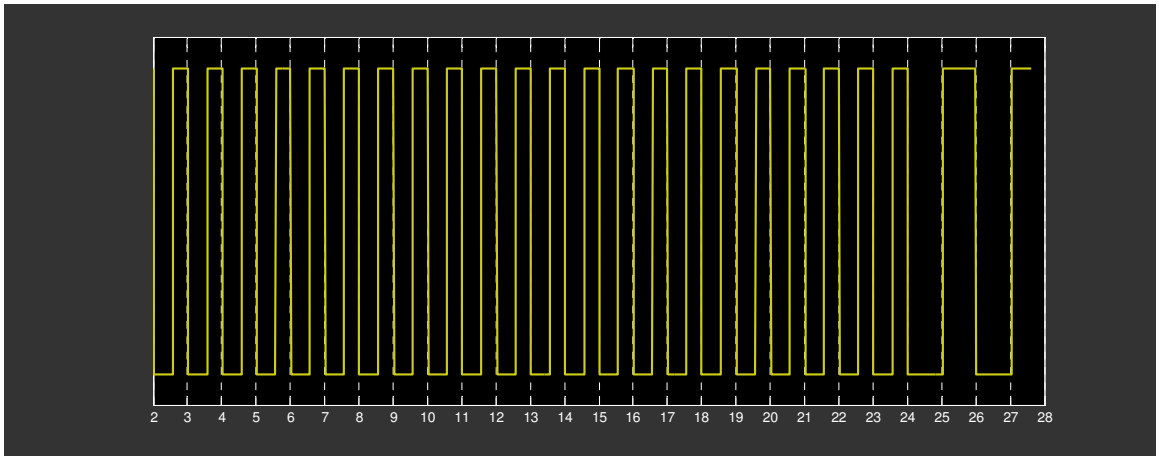

Figure 9. Version 3.0 Preamble

This also illustrates that for each clock period containing a "1" bit in the preamble, there are two short periods – a short RF off period followed by a short RF on period. This illustrates that it requires two short intervals to transmit a bit of the same value as the previous bit. This is true whether the previous bit is a "0" or a "1".

The preamble is 24-bits long, and after the 24th bit a long off period is generated. This is the beginning of the sync nibble.

The next figure shows a zoomed-in view of the sync nibble. At the clock transition labeled zero, the RF is off prior to the transition. This indicates a zero bit. Each of the next three transitions (1,2,3) show the bit being flipped from the previous transition so the 4-bit sync nibble is “0101” in the order of transmission. If we take the sync nibble in the opposite order (“1010”) it becomes a hexadecimal “A”.

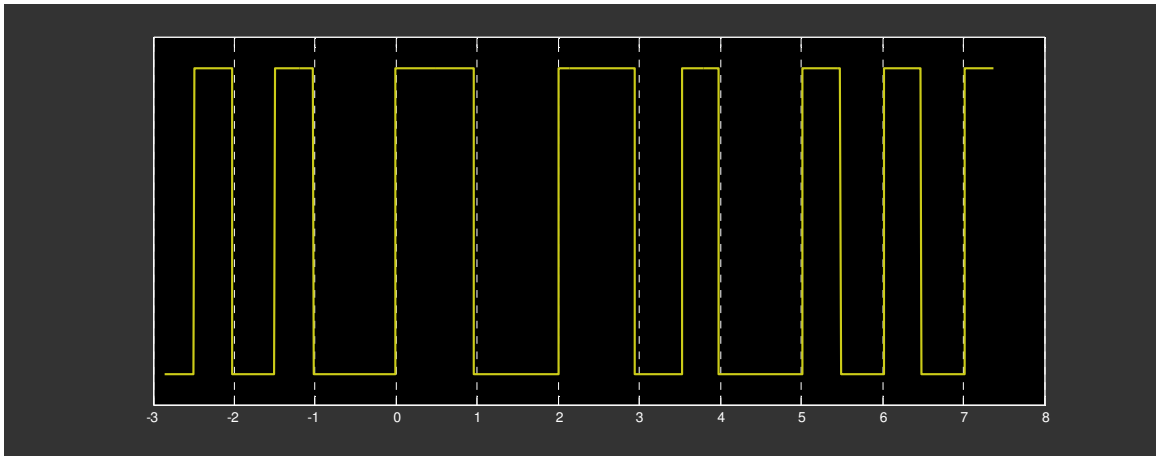

Figure 10. Version 3.0 Sync Nibble

The sync nibble also demonstrates that in order to send a bit which is the opposite of the previous bit, a long off or on period is generated.

This is good point to review the two algorithms for decoding. In the first case, we simply use the state of the RF signal (on/off) prior to the middle of the clock period to decode the bit. In figure 4, the horizontal axis grid lines are aligned with the middle of each clock period. By inspection, the bit sequence here (starting at “-2”) is 1,1,0,1,0,1,1,0,0,0.

In the second algorithm, we start out with the knowledge that the preamble contains all “1” bits. Further knowledge of RF state at transition points is not used in this algorithm. When the first long period is detected we have reached the end of the preamble.

Remember that long periods signal a bit which is opposite from the previous bit and the preamble contains all “1” bits. Therefore the first long period signals a “0” bit. Likewise, the next long period signals a “1” bit since the preceding bit was a “0”.

Preamble bits are present at clock transitions labeled (0,1,2,3) so the transition labeled “4” is the first data bit. First, we can clearly see that this bit is a “1” since the RF was “on” just prior to the transition at “4”. However, we

also know this is a “1” because the last sync bit was a “1” and this was followed by two short periods. Remember that short periods signal a bit which is identical to the previous bit.

Now we’ll take a look at a longer segment of a version 3.0 message. The transition corresponding to the first sync bit is labeled “0”. Using our time-based decoding algorithm, we classify the clock intervals starting at “0” as either containing one long period “L” (either on or off), or two short periods “S” (either on-to-off or off-to-on). By inspection the following sequence results.

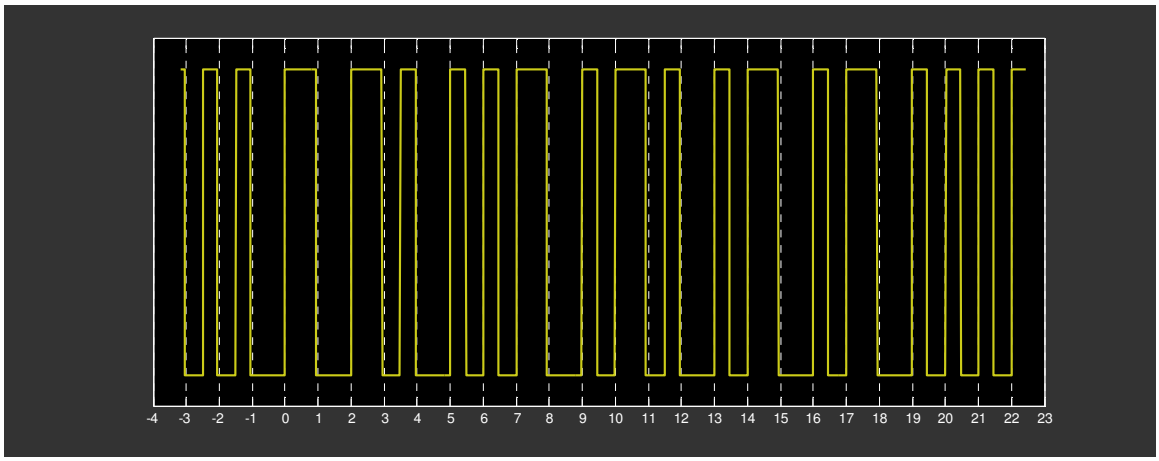

Figure 11. Version 3.0 Data Segment

```
L L L L   S L S S   L L S L   S L S L   L S L L   S S S
0 1 0 1   1 0 0 0   1 0 0 1   1 0 0 1   0 0 1 0   0 0 0

    A         1         9         9         4
```

Adding the knowledge that the first sync bit is a zero, we can now decode the bit stream by inspection – writing down the same bit for “S” and the opposite bit for “L”.

The next step is to group the bits into nibbles and reverse the order of the bits in each nibble. This gives us the hexadecimal sequence shown above. The first four nibbles (hexadecimal “1994”) are the ID code for the WGR800 anemometer.

The next figure shows what happens at the end of a version 3.0 RF message. At the clock transition labeled “0”, the RF signal simply goes off, and stays off. We will hear no more from this sensor for about another minute.

After the RF signal has been off for perhaps three or so clock periods, the receiver begins to crank up its internal gain. This is controlled by the receiver’s automatic gain control (or AGC) circuit. After a few more clock periods (between 5 and 6 on the horizontal axis), the gain has been increased so much

that low level RF noise is now being mistakenly detected as an RF signal going on and off.

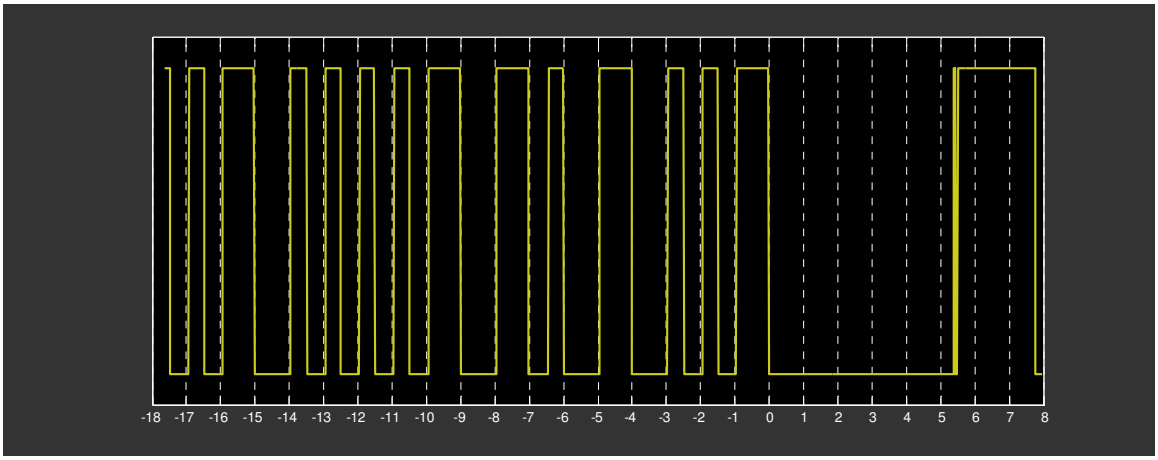

Figure 12. End of a Version 3.0 Message

Notice that the end of the signal is followed by an off period that lasts a little over five clock periods. This is an invalid length for a measured time interval so we can use this to identify the end of the message.

The on and off periods generated by noise will generally not be of a length we would consider to be valid “on” or “off” time intervals. As a result there is about zero chance we will mistake this noise output for a valid sensor message. Once in a while, a small number of time periods (maybe one to three or so) will occur that fall within the expected limits but we are looking for many more than this in sequence to identify a valid preamble.

Version 2.1 Protocol Samples

Here’s what a version 2.1 protocol preamble looks like. Notice that it contains 32 bits, all of which are long time intervals (on and off). Based on our interval-based decoding algorithm, we know then that the preamble contains an alternating sequence of “1” and “0” bits.

The last preamble bit (at the “0” clock transition) is a “1” bit since the RF signal is on just prior to the transition. The RF pulse that ends at clock transition “-32” is not part of the preamble; it just exists to wake up the receiver and allow time for the AGC circuit to get adjusted. The first actual preamble bit is the “0” that occurs at clock transition “-31”.

The preamble is then a sequence of 32 alternating bits: “010101...0101”. Now, recall that version 2.1 messages actually send each bit twice, with the first of the two inverted. Therefore, a sequence of sixteen “1” bits will be sent as 32 bits and each original “1” bit is sent as a “01” pair of bits. This yields the

actual preamble we see here, so in fact the original preamble is of length 16 and is all “1” bits.

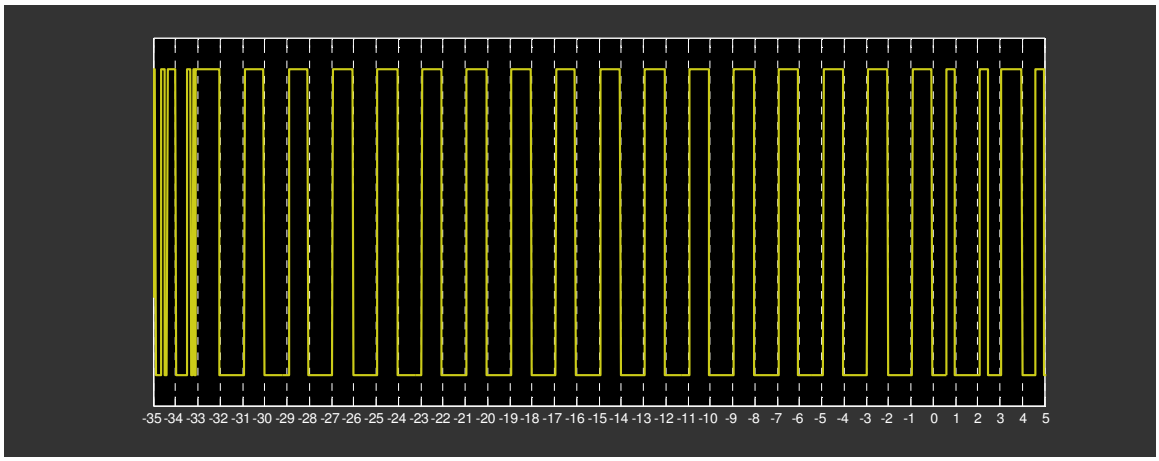

Figure 13. Version 2.1 Preamble

Because of this doubling of bits, we’ll refer to each original bit as a “bit pair”. Each pair is either a “10” or a “01” – “00” and “11” are not legal bit pairs.

Now, take a look at the sync nibble following the preamble. In the next figure, the last bit of the last preamble bit pair occurs at clock transition “0”. The first bit of the first sync bit pair then occurs at transition “1”. Also, remember that the preamble sequence ends with a “01” pair.

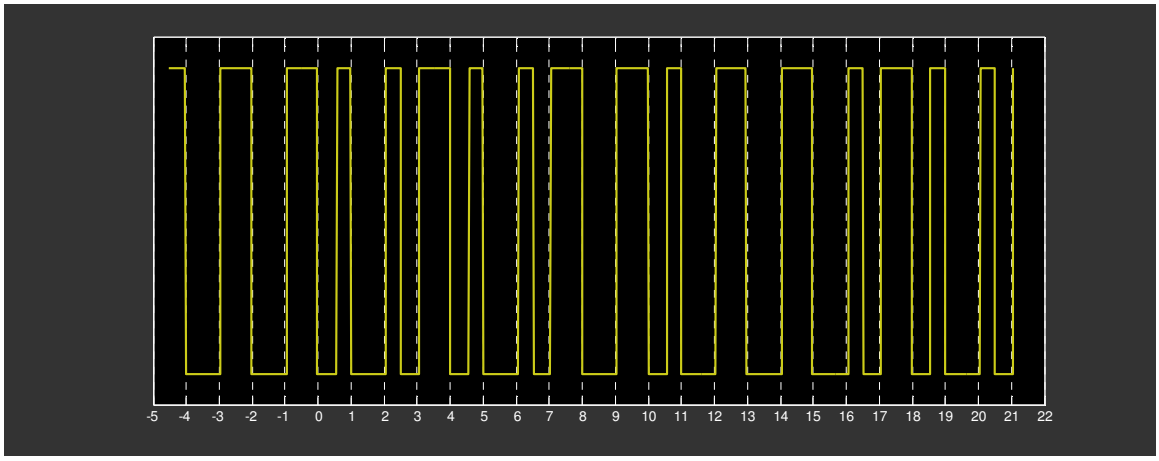

Figure 14. Version 2.1 Sync and Data

SL	SL	SL	SL		LL	SL	LL	LL		SL	SL
10	01	10	01		01	10	10	10		01	10
0	1	0	1		1	0	0	0		1	0
		A					1				

The first bit pair of the sync nibble contains a two short periods followed by a long period (“SL”). Since the last preamble bit was “1”, the “SL” sequence represents a “10” bit pair. The original sync bit is equal to the last bit in the pair and is therefore a “0”.

Then next bit pair (between 2-3 and 3-4 in the above figure) is a “SL” sequence again. Since the last bit of the previous pair was a “0”, this sequence is a “01”, corresponding to an original bit value of “1”.

The short/long periods are grouped into pairs above so the bit pairs are easily seen. Since the second bit of each pair is not inverted from the original message bit, we extract them to get the original message bits.

It is fairly obvious now that since the second bit in each pair is opposite of the bit preceding it, a long interval (RF on or off) is required to transmit the second bit. The net result is that each bit pair is either going to be “SL” or “LL”. A bit pair of “SS” or “LS” is illegal since the second bit in this case would be identical to the preceding bit.

Since there are twice as many bits, this example only shows the sync nibble and the next six bits.

The following plot shows one of the possible endings for a version 2.1 RF message. However it doesn’t actually end at this point. In addition to containing two bits for every original message bit, the entire RF message is actually sent twice.

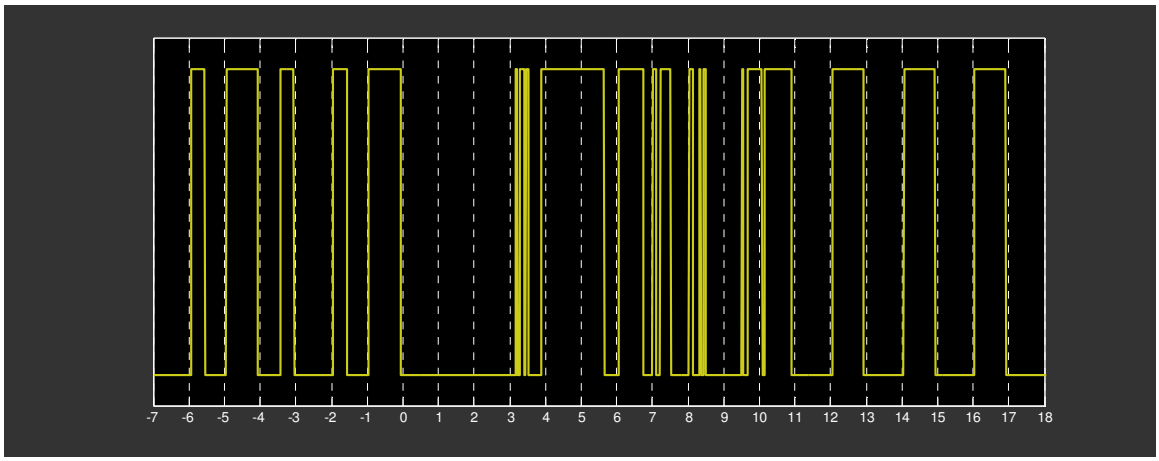

Figure 15. Version 2.1 Double Message

Above, the first copy of the message ends at clock transition “0”. The receiver starts detecting noise just after transition “3”. Then, at transition “10” the second copy of the message appears. This is an exact copy of the first message – preamble, sync, data and all.

In the above example there are ten clock periods where the RF is off and before the second copy of the message begins. This is typical from sensors such as the THGR122NX. However, this is not always the case as is shown in the next figure.

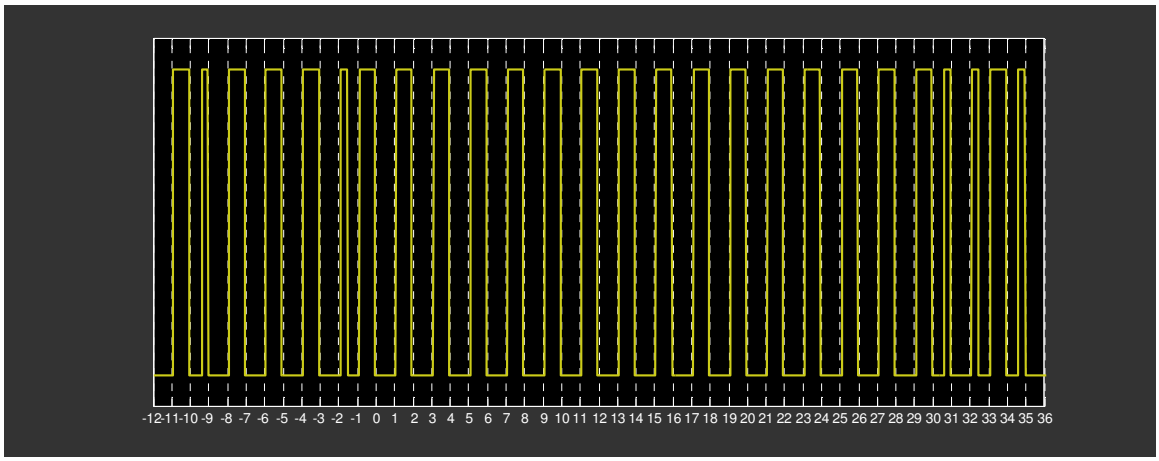

Figure 16. Version 2.1 Double Message w/o Pause

In this example (from a UVR128 UV sensor) the first message ends at clock transition “-2”. The preamble for the second copy of the message begins immediately without any pause at all. The first “01” bit pair of the next preamble occurs at transitions -1 and 0.

For this scenario, the decoding algorithm will simply continue to collect valid message bits until the second copy of the message ends and the receiver starts decoding noise. Once the bits are decoded, we must look for a sequence of sixteen “1” bits in the middle of the message to find the second copy.

Version 1.0 Protocol Samples

The version 1.0 preamble is shown below. The clock rate used to generate the x-axis was 342Hz. The integer values on the horizontal axis are aligned with the middle of each clock period.

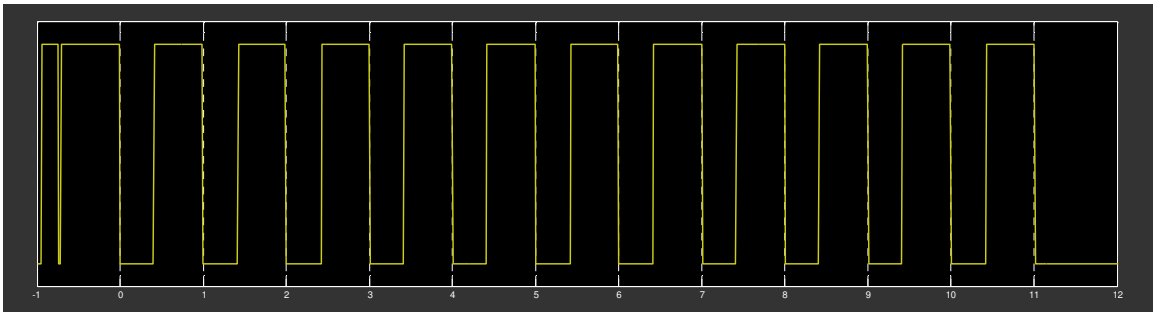

Figure 17. Version 1.0 RF Preamble

Since the transmitted bit is equal to the RF state just before the middle of the clock period, this preamble consists of 12 “1” bits. These occur starting at zero on the labeled plot, and the transition defining the last preamble bit is at eleven.

The next graphic shows the sync portion of the RF message. The middle of the first clock period after the preamble is numbered “12” in this graphic. The sync interval runs from clock periods 12 through 15 in this case.

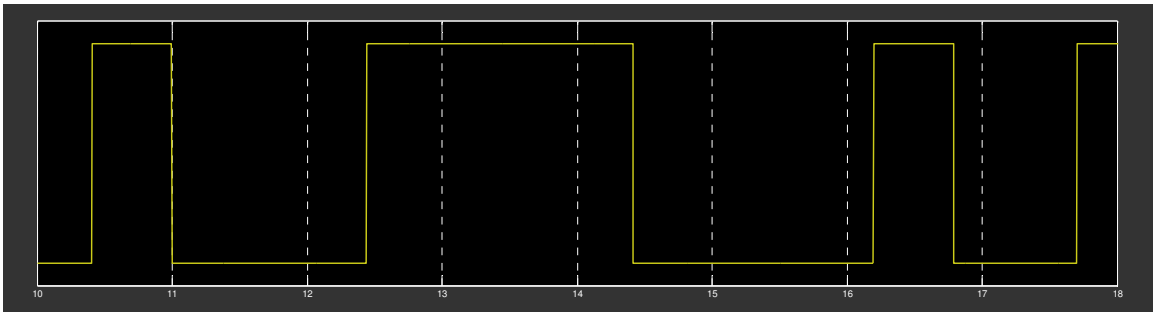

Figure 18. Version 1.0 Sync Interval

By the standards used in the rest of the RF message, this sync period is illegal because it has no RF transitions in the middle of each clock period. However, if we continue to sample the RF state just before the middle of each clock period, the sync portion of the message contains five bits – 0,1,1,0,0.

Clock alignment jumps slightly between the end of the sync period and the first data bit. Above, the transition which occurs just prior to the middle of clock period 17 is actually the middle of the first data clock period. It is not known why this apparent time shift exists.

The next figure shows the data portion of the message after the clock has been re-synchronized after the sync period. The middle of the clock period containing the first data bit is numbered zero.

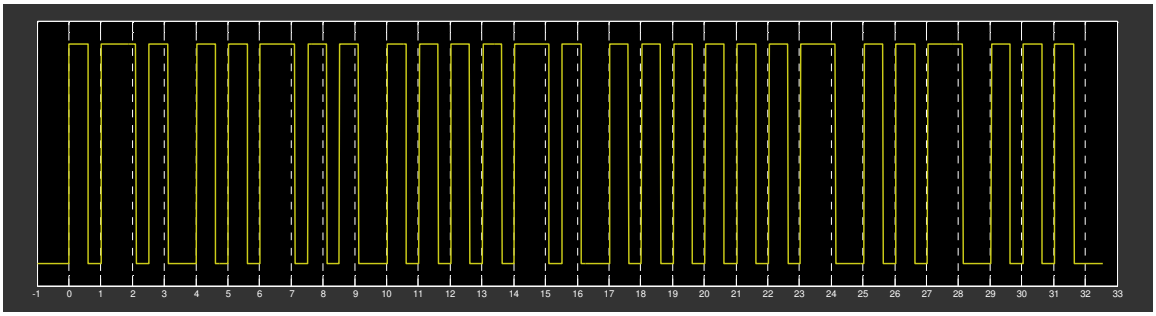

Figure 19. Version 1.0 Data Payload (starting with “0”)

Recall that the RF pulse is off at the end of the sync period. There are two ways the data portion of the message can begin depending on the value of the first data bit.

If that bit is a zero, then the RF will remain off until the middle of the first clock period. Since there must be a transition in the middle of the clock period, the RF will need to go on at that point. This is the situation seen in figure 14. Obviously, that first pulse can either be long or short depending on the value of the second bit. In this case, the second bit is also zero so the pulse is a short one.

The next graphic shows the case where the first data bit is a “1”. In this case there is a transition prior to the middle of the first clock period. Since a transition is required at the middle of the clock period, this pulse must be a short one.

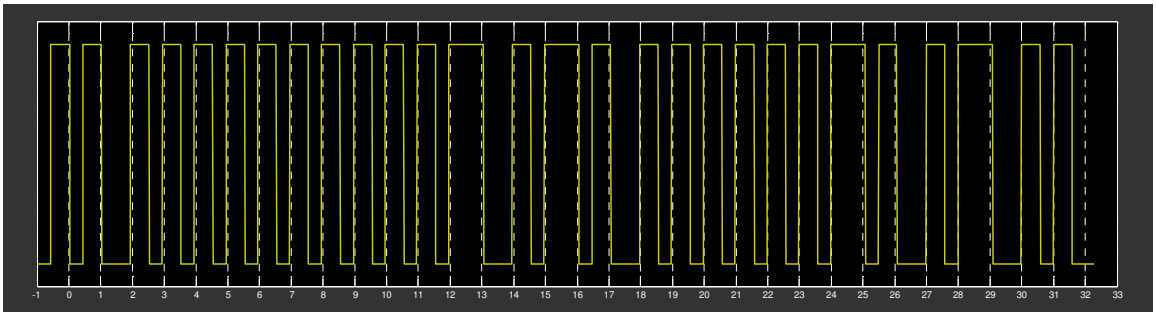

Figure 20. Version 1.0 Data Payload (starting with "1")

Referring back to figure 13, it is clear that the length of the off period after the long sync pulse can have two different values. The shorter value (shown in figure 13) occurs when the first data bit is a “1” and the longer value corresponds to a first data bit of “0”.

As mentioned above, and for unknown reasons, a clock synchronized with the preamble is slightly out of sync with the data portion of the message. The measured time from the end of the long sync pulse to the middle of the first data clock period is 6.68 milliseconds.

AGC Problems

In some receivers, the long RF-off time periods that occur during the sync interval of version 1.0 RF messages may cause problems with automatic gain control. If the receiver is designed to receive data at kilo-hertz rates, the AGC may start ramping up receiver gains during these long RF-off intervals. When the RF finally comes back on, the receiver may be over-loaded and the first few data bits will be corrupted until the AGC can recover.

This problem can be solved if the AGC circuits are locked down (frozen) at some point during the preamble of a version 1.0 RF message and unlocked after the message ends. This is the technique used with the WSDL WxShield to receive version 1.0 messages.

Appendix II

CRC Specifics

When the OS post-amble contains an 8-bit CRC checksum, it is a CRC-8-CCITT code, which uses the polynomial:

$$x^8 + x^2 + x^1 + x^0$$

The CRC checksum is transmitted as two nibbles, least significant nibble first. The bit-stream fed to the CRC algorithm is created by sending each nibble, MSB first, in the order of transmission up to, but not including the checksum. Note that bits within a nibble are ***not*** fed to the CRC algorithm in the order of transmission (which is LSB first). For example, a the hexadecimal sensor ID "2914" is transmitted as this bit sequence (LSB of each nibble goes out first):

0100 1001 1000 0010

However, this portion of the message would fed to the CRC algorithm in this order (MSB of each nibble goes out first):

0010 1001 0001 0100

Further details appear to be sensor/protocol specific. The simplest case is found with version 3.0 sensors. These use an initial shift register value of zero.

Version 2.1 sensors appear to use different initial register values -- even among sensors of the same model. It is not clear if this is actually correct -- it seems a little odd to ask the receiver of these messages to figure out the correct initial register value. Perhaps there is a simpler explanation for the CRC algorithm used in version 2.1 sensors.

Ambient Weather WH2C sensors also generate an 8-bit CRC using the Dallas/Maxim polynomial:

$$x^8 + x^5 + x^4 + x^0$$