Reverse Engineering Acurite Temperature Sensor

- Overview
- Specifications
- RTL_433 Reverse Engineering
- What is an OOK package?
- Finding the Bits
- The Protocol
    - Details
    - Checksum/Hash
- Sending Temperatures to the Receiver
    - Transmitter Code
- Reading the values from the Transmitter
    - Receiver Code
- Presentation
- References

Overview

Specifications

Product:

Info	URL
Website	https://www.acurite.com/kbase/thermometers/00606_Thermometer.html
Manual	https://www.acurite.com/media/manuals/00606-instructions.pdf

Transmitter:

Specification	Value
Model	00606TXA1
FCC ID	RNE606TXA1

Receiver:

Specification	Value
Model	00606TXA1
FCC ID	RNE606TXA1

RTL_433 Reverse Engineering

Using rtl_433:

> rtl_433

Output:

```
rtl_433 version 19.08-18-g8eecdbb branch master at 201909241811 inputs file rtl_tcp RTL-SDR
Use -h for usage help and see https://triq.org/ for documentation.
Trying conf file at "rtl_433.conf"...
Trying conf file at "/Users/john.mehan/.config/rtl_433/rtl_433.conf"...
Trying conf file at "/usr/local/etc/rtl_433/rtl_433.conf"...
Trying conf file at "/etc/rtl_433/rtl_433.conf"...

Consider using "-M newmodel" to transition to new model keys. This will become the default someday.
A table of changes and discussion is at https://github.com/merbanan/rtl_433/pull/986.

Registered 108 out of 138 device decoding protocols [ 1-4 8 11-12 15-17 19-21 23 25-26 29-36 38-60 63 67-71 73-
100 102-103 108-116 119 121 124-128 131-138 ]
Found Rafael Micro R820T tuner
Exact sample rate is: 250000.000414 Hz
[R82XX] PLL not locked!
Sample rate set to 250000 S/s.
Tuner gain set to Auto.
Tuned to 433.920MHz.
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time : 2019-10-03 11:08:26
model : Acurite 606TX Sensor id : 120
Battery : OK Temperature: 25.9 C Integrity : CHECKSUM
```

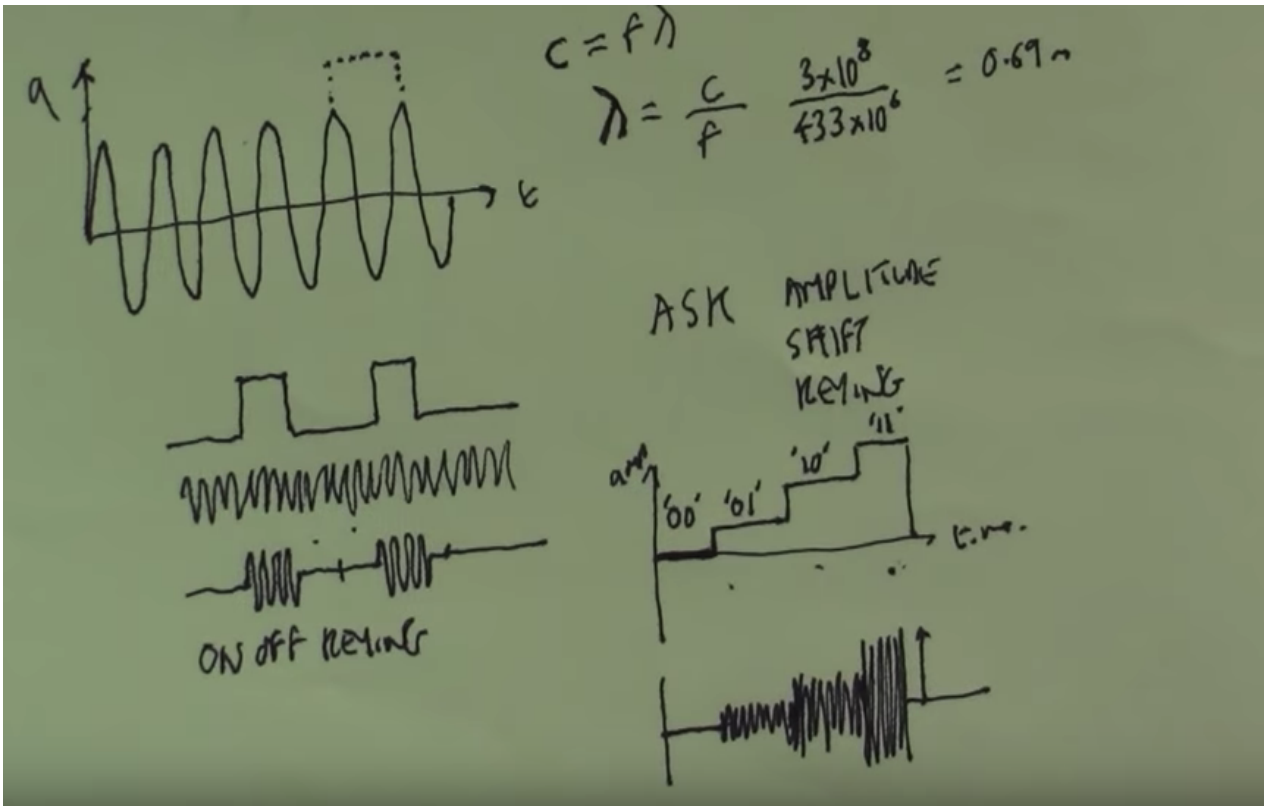
> rtl

```
Detected OOK package        2019-10-03 11:49:46
Analyzing pulses...
Total count:  237,  width: 744.45 ms               (186112 S)
Pulse width distribution:
 [ 0] count:  237,  width:  508 us [496;516]       ( 127 S)
Gap width distribution:
 [ 0] count:  154,  width: 1932 us [1924;1944]       ( 483 S)
 [ 1] count:   70,  width: 3884 us [3876;3896]       ( 971 S)
 [ 2] count:    5,  width:  468 us [460;476]       ( 117 S)
 [ 3] count:    6,  width: 8520 us [8516;8532]       (2130 S)
 [ 4] count:    1,  width:  708 us [708;708]       ( 177 S)
Pulse period distribution:
 [ 0] count:  154,  width: 2440 us [2432;2456]       ( 610 S)
 [ 1] count:   70,  width: 4392 us [4380;4408]       (1098 S)
 [ 2] count:    5,  width:  972 us [972;980]       ( 243 S)
 [ 3] count:    6,  width: 9032 us [9028;9040]       (2258 S)
 [ 4] count:    1,  width: 1220 us [1220;1220]       ( 305 S)
Level estimates [high, low]:  15939,   1378
RSSI: -0.1 dB SNR: 10.6 dB Noise: -10.7 dB
Frequency offsets [F1, F2]:   18554,      0       (+70.8 kHz, +0.0 kHz)
Guessing modulation: Pulse Position Modulation with fixed pulse width
Attempting demodulation... short_width: 468, long_width: 708, reset_limit: 8536, sync_width: 0
Use a flex decoder with -X 'n=name,m=OOK_PPM,s=468,l=708,g=712,r=8536'
pulse_demod_ppm(): Analyzer Device
bitbuffer:: Number of rows: 25
[00] { 0}           :
[01] { 0}           :
[02] { 0}           :
[03] { 0}           :
[04] { 0}           :
[05] { 0}           :
[06] { 0}           :
[07] { 0}           :
[08] { 0}           :
[09] { 0}           :
[10] { 0}           :
[11] { 0}           :
[12] { 0}           :
[13] { 0}           :
[14] { 0}           :
[15] { 0}           :
[16] { 0}           :
[17] { 0}           :
[18] { 0}           :
[19] { 0}           :
[20] { 0}           :
[21] { 0}           :
[22] { 0}           :
[23] { 0}           :
[24] { 0}           :
... Maximum number of rows reached. Message is likely truncated.
```

So, this is showing us that it detected OOK modulation but when it tries to convert to bits, it fails...

What is an OOK package?

On Off Keying is a form of Amplitude Shift Keying (ASK) limited to two values. If you use ASK with only two values, you are essentially using OOK.



Finding the Bits

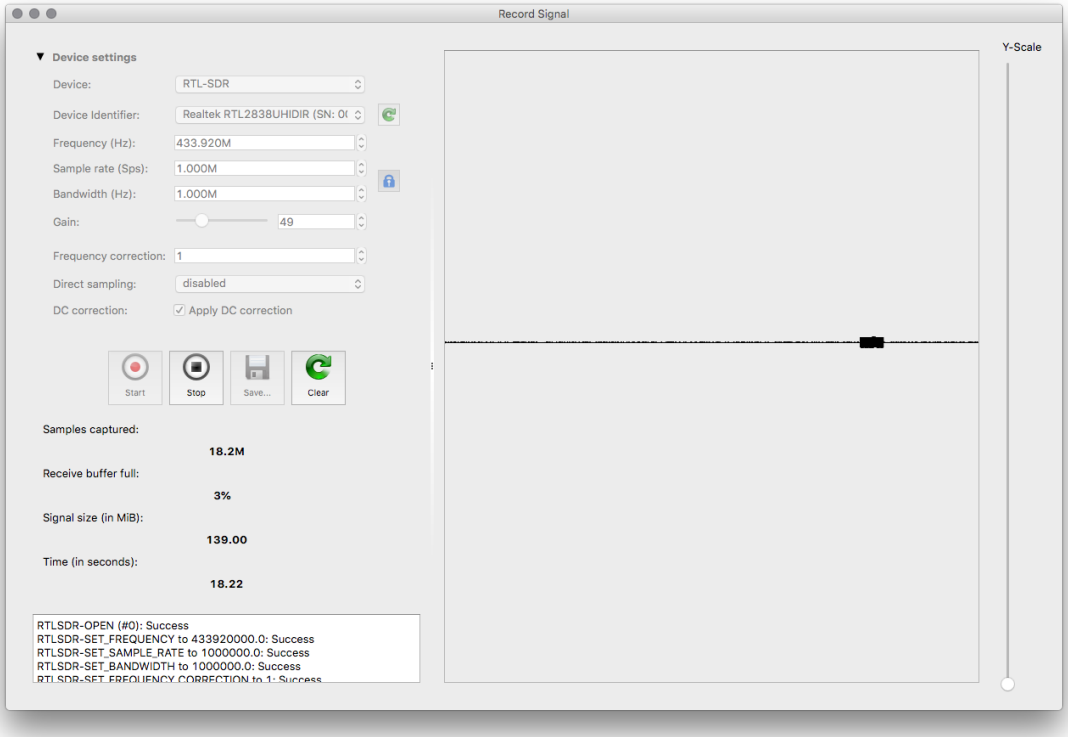
Install Let's install **Universal Radio Hacker**.

> pip3 install urh

Running URH

> urh

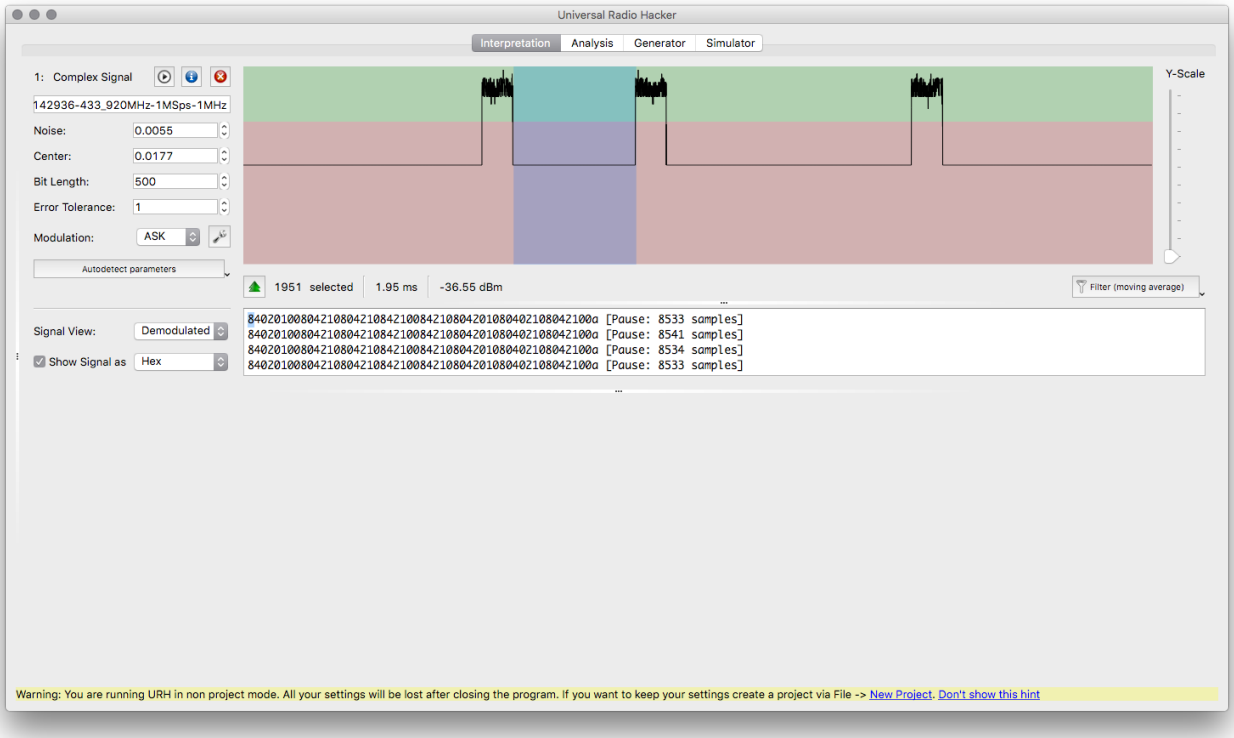
From the initial window, select File Record Signal



Click Start button and once you noticed your signal, click Stop and Save. Close the window

The Interpretation Screen should appear.

- zoom in on the first burst.
- Set the signal view to Demodulated.



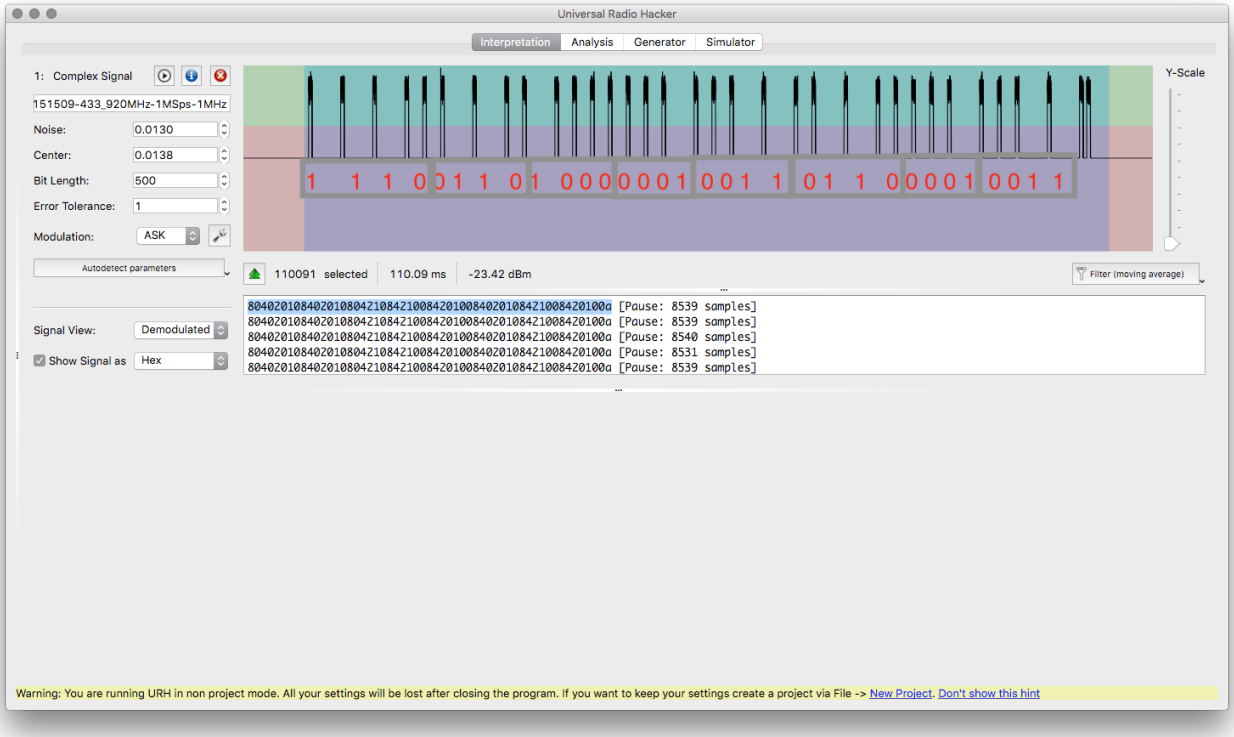
Looking at the entire signal, we see that it sends the identical information 7 times.

This particular signal starts every bit by turning ON the RF signal and then turning if off for either 2 or 4 milliseconds. Staying off for 2 ms signifies a value of 0 while staying off for 4 ms signifies a value of 1.

What we deduced from looking at the signal:

Pulse	Delay
On Pulse	0.5 ms
0 value	2 ms off between On Pulse
1 value	4 ms off between On Pulse
Stop Pulse	0.5 ms
	9 ms delay between repeats repeat the whole thing a total of 7 times.

If we look at just one of the pulses, we see that it contains 32 bits or 4 bytes:



We received the following data:

1110 0110 1000 0001 0011 0110 0001 0011 (E6 81 36 13 hex)

The Protocol

The protocol for this device is publish at the following URL:

http://www.osengr.org/WxShield/Downloads/Weather-Sensor-RF-Protocols.pdf

Details

The protocol for the AcuRite 00606TX is:

All in nibbles (4 bits).

Rolling Code(2), Status(1), Temperature (signed binary), Hash Code

Data being sent:

Nible	Purpose	Our Value	Description
1-2	Rolling Code	E6	The rolling code randomly generated when the transmitter powers up. After the receiver receives it's first rolling code, it will not accept messages with a different code unless rebooted.
3	Status	8	A status of 8 signifies that the batter is ok. 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.
4-6	Temperature	136	The temperature with 1 decimal place. A temperature of 31.0C = 31.0*10 = 310 dec = 136 hex
7-8	Hash	13	Hash computed using

Checksum/Hash

> *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.*

Algorithm

1. Generate LSFR sequence equal to the length of the message.

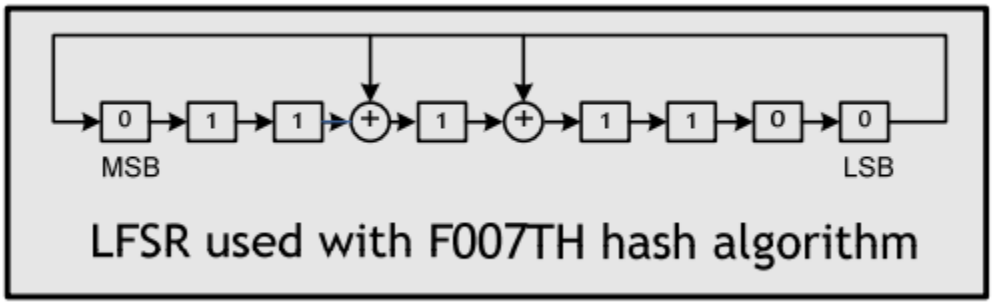
In our case, our message is 24 bits long, so we will need 24 values plus an addition 5 since we are going to start at the fifth byte in the sequence.

- Start with an 8-bit register initialized to the value 0x7C (0111 1100)
- 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 32 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.



	Start Value	Shift Right	XOR with 0x18 if MSB =1 (flip bits 3 and 4)	Xor	Value
1	0111 1100 (0x7C)	0011 1110	NO	-	0011 1110 (0x3e)
2	0011 1110 (0x3e)	0001 1111	NO	-	0001 1111 (0x1f)
3	0001 1111 (0x1f)	1000 1111	YES	xor 0x18	1001 0111 (0x97)
4	1001 0111 (0x97)	1100 1011	YES	xor 0x18	1101 0011 (0xd3)
5	1101 0011 (0xd3)	1110 1001	YES	xor 0x18	1111 0001 (0xf1)
...					
32	1000 0110 (0x86)	0100 0011	NO	-	0100 0011 (0x43)
...					
37	0011 0001 (0x31)	1001 1000	YES	xor 0x18	1000 000 (0x80)

Values

```
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 ..
```

2. Combines the LSFR sequence with message bits to form the final hash value

> *To compute the message hash value, sequence through the 24 message bits in the order they were received. Since everything here is big-endian, that means proceeding from MSB to LSB. Start by initializing the hash register to the value 0x00. As the message bits are read, for every bit in the message that is a one, exclusive-or the corresponding value (+5) from the LSFR sequence into the hash register.*
>
> *For example, if the 10th bit is a one, then take the 15th 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 0x00.*

From MSB to LSB for message 1110 0110 1000 0001 0011 0110 (E6 81 36 hex)

Register Value	Index	Value	Value=1?	Calculation	New Register Value (xor with LSFR Sequence starting at the 5th (0xF1)
00	23 -MSB	1	Yes	00 xor f1	f1
f1	22	1	Yes	f1 xor e0	11
11	21	1	Yes	11 xor 70	61
...					

13	0	0	No	-	13

Checksum/Hash Source Code

```
#define MESSAGE_SIZE 32
#define OFFSET 4

uint8_t LSFR_sequence[MESSAGE_SIZE] = {0};

void calculateLSFR() {
        int i;
        uint8_t reg = 0x7C;
        uint8_t temp_reg = 0;

        for (i = 0; i < MESSAGE_SIZE; i++) {
                temp_reg = reg & 0x01;
                reg >>= 1;
                reg |= (temp_reg << 7);

                if (temp_reg) {
                        reg ^= 0x18;
                }

                LSFR_sequence[i] = reg;
                //printf("%02x\n", LSFR_sequence[i]);
        }
}

uint8_t combineLSFR(uint8_t len, uint8_t *data) {
        uint8_t hash_reg = 0; // not 0x64
        int byte_idx, bit_idx;
        uint8_t byte, bit;
        //printf("***COMBINE\n");

        for (byte_idx = 0; byte_idx < len; byte_idx++) {
                for (bit_idx = 7; bit_idx >= 0; bit_idx--) {
                        bit = (data[byte_idx] & (1 << bit_idx)) >> bit_idx;
                        if (bit) {
                                hash_reg ^= LSFR_sequence[byte_idx * 8 + (7 - bit_idx) + OFFSET];
                                //printf("[%d]: %02x\n", byte_idx * 8 + (7 - bit_idx), hash_reg);
                        }
                        bit = 0;
                }
        }

        return hash_reg;
}

uint8_t Checksum(int length, uint8_t *buff) {
        calculateLSFR();
        return combineLSFR(length, buff);
}
```

Sending Temperatures to the Receiver

In an effort to send our own temperatures to the receiver, we put together a program running on an ESP8266 (Wemos mini) connected with RF transmitter.

Transmitter Code

```
// ----------------------------------------------------------------------------
// Acurite 00606TX Transmitter
// ----------------------------------------------------------------------------
```

```
#define SYNC_LENGTH  9000
#define SEP_LENGTH   500
#define BIT1_LENGTH  4000
#define BIT0_LENGTH  2000

#define MESSAGE_SIZE 32
#define LSFR_OFFSET 4

uint8_t LSFR_sequence[MESSAGE_SIZE] = {0};

const int TX_PIN = D2;
byte payload[4];

// ----------------------------------------------------------------------------
// setup
// ----------------------------------------------------------------------------
void setup ( void ) {
    pinMode(TX_PIN, OUTPUT);
    Serial.begin ( 115200 );
    Serial.println("Sending...");
    generatePayload();
    doTransmission();
}
// ----------------------------------------------------------------------------
// loop
// ----------------------------------------------------------------------------
void loop ( void ) {
  delay(20000);
}

// ----------------------------------------------------------------------------
// generatePayload
// ----------------------------------------------------------------------------
void generatePayload(){
      //define rolling code
    byte rollingCode = 0xAA;

    //generate random temperature
    int temperature = (int)random(-400,400);
    Serial.printf("Temp: %.1f\n",temperature/10.0);

    unsigned int hightByteTemp = temperature >> 8;
    unsigned int lowByteTemp = temperature & 0xff;

    //add battery OK flag to
    hightByteTemp = 0x80 + (hightByteTemp & 0x0F);

    payload[0] = rollingCode;
    payload[1] = hightByteTemp;
    payload[2] = lowByteTemp;
    payload[3] = computeChecksum(3,payload);

    Serial.print("Payload: ");
    for(int i=0;i<4;i++){
      Serial.printf("%02x ",payload[i]);
    }
    Serial.println("");
}

// ----------------------------------------------------------------------------
// toggleLED
// ----------------------------------------------------------------------------
void toggleLED(){
    //flash led
    digitalWrite(LED_BUILTIN, HIGH);
    delay(500);
    digitalWrite(LED_BUILTIN, LOW);
}

// ----------------------------------------------------------------------------
// doTransmission
```

```
// ----------------------------------------------------------------------------------------
void doTransmission() {

    Serial.println("Doing transmission...");
    toggleLED();

    for (int burst = 1; burst <= 7; burst++) {
      Serial.print("Sending: ");
      sendPayload();
      sendSync();
      Serial.println("");
    }
}

// ----------------------------------------------------------------------------------------
// sendPayload
// ----------------------------------------------------------------------------------------
void sendPayload(){
  for(int i=0;i<4;i++){
    sendByte(payload[i]);
  }
}

// ----------------------------------------------------------------------------------------
// sendByte
// ----------------------------------------------------------------------------------------
void sendByte(byte b){
  for(int i=7;i>=0;i--){
    int bitValue = readBit(b,i);
    sendBit(bitValue);
    if(i%4==0){
      Serial.print(" ");
    }
  }
}

// ----------------------------------------------------------------------------------------
// readBit
// ----------------------------------------------------------------------------------------
int readBit(byte b, int bitPos){
  int x = b & (1 << bitPos);
  return x == 0 ? 0 : 1;
}

// ----------------------------------------------------------------------------------------
// sendBit
// ----------------------------------------------------------------------------------------
void sendBit(int val){

  digitalWrite(TX_PIN, HIGH );
  delayMicroseconds(SEP_LENGTH );

  digitalWrite(TX_PIN, LOW );
  if(val==0){
    delayMicroseconds(BIT0_LENGTH );
    Serial.print("0");
  }else{
    delayMicroseconds(BIT1_LENGTH );
    Serial.print("1");
  }
}

// ----------------------------------------------------------------------------------------
// sendSync
// ----------------------------------------------------------------------------------------
void sendSync(){
  digitalWrite(TX_PIN, HIGH );
  delayMicroseconds(SEP_LENGTH);
  digitalWrite(TX_PIN, LOW );
  delayMicroseconds(SYNC_LENGTH );
  Serial.print(" -");
```

```
}

// ----------------------------------------------------------------------------------------
// calculateLSFR
// ----------------------------------------------------------------------------------------
void calculateLSFR() {
    int i;
    uint8_t reg = 0x7C;
    uint8_t temp_reg = 0;

    for (i = 0; i < MESSAGE_SIZE; i++) {
        temp_reg = reg & 0x01;
        reg >>= 1;
        reg |= (temp_reg << 7);

        if (temp_reg) {
            reg ^= 0x18;
        }

        LSFR_sequence[i] = reg;
        //printf("%02x\n", LSFR_sequence[i]);
    }
}
// ----------------------------------------------------------------------------------------
// combineLSFR
// ----------------------------------------------------------------------------------------
uint8_t combineLSFR(uint8_t len, uint8_t *data) {
    uint8_t hash_reg = 0; // not 0x64
    int byte_idx, bit_idx;
    uint8_t byte, bit;
    //printf("***COMBINE\n");

    for (byte_idx = 0; byte_idx < len; byte_idx++) {
        for (bit_idx = 7; bit_idx >= 0; bit_idx--) {
            bit = (data[byte_idx] & (1 << bit_idx)) >> bit_idx;
            if (bit) {
                hash_reg ^= LSFR_sequence[byte_idx * 8 + (7 - bit_idx) + LSFR_OFFSET];
                //printf("[%d]: %02x\n", byte_idx * 8 + (7 - bit_idx), hash_reg);
            }
            bit = 0;
        }
    }

    return hash_reg;
}

// ----------------------------------------------------------------------------------------
// computeChecksum
// ----------------------------------------------------------------------------------------
uint8_t computeChecksum(int length, uint8_t *buff) {
    calculateLSFR();
    return combineLSFR(length, buff);
}
```

Reading the values from the Transmitter

Receiver Code

```
// ----------------------------------------------------------------------------------------
// Acurite 00606TX Reader
// Adapted from code by Ray Wang (Rayshobby LLC)
// ----------------------------------------------------------------------------------------

#define DATAPIN  D2
```

```
// ring buffer size has to be large enough to fit
// data between two successive sync signals
#define RING_BUFFER_SIZE  256

#define SYNC_LENGTH  9000
#define SEP_LENGTH   500
#define BIT1_LENGTH  4000
#define BIT0_LENGTH  2000

#define MESSAGE_SIZE 32
#define LSFR_OFFSET 4

uint8_t LSFR_sequence[MESSAGE_SIZE] = {0};

//ringbuffer contains the time since the last signal was off.
unsigned long timings[RING_BUFFER_SIZE];

unsigned int syncIndex1 = 0;  // index of the first sync signal
unsigned int syncIndex2 = 0;  // index of the second sync signal
bool received = false;

//protypes
void ICACHE_RAM_ATTR handler();

// ----------------------------------------------------------------------------------------
// setup
// ----------------------------------------------------------------------------------------
void setup() {
  Serial.begin(115200);
  pinMode(DATAPIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(DATAPIN), handler, CHANGE);
  Serial.println("\n\nStarted");
}

// ----------------------------------------------------------------------------------------
// loop
// ----------------------------------------------------------------------------------------
void loop() {

  //Did we get a message?
  if (received == true) {
    Serial.println("\nReceived...");

    // disable interrupt to avoid new data corrupting the buffer
    detachInterrupt(digitalPinToInterrupt(DATAPIN));

    //processSignal
    unsigned int bits[34];
    bool error = processSignal(bits);

   if(error==false){

      //convert our bits to byte array
      byte data[4];
      bool success = bitsToByteArray(bits,32,data,4);

      if(success){
        byte rollingCode = data[0];
        Serial.printf("Rolling Code: 0x%02x\n",rollingCode);

        byte batteryStatus = (data[1] & 0xf0) >> 4;
        Serial.printf("Battery Status: 0x%02x\n",batteryStatus);

        float temperature = computeTemperature(data[1],data[2]);
        Serial.printf("Temp: %0.1f C\n",temperature);

        byte checksum = data[3];
        Serial.printf("checksum: 0x%02x\n",checksum);

        //uint8_t computeChecksum(int length, uint8_t *buff) {
        byte computedChecksum = computeChecksum(3,data);
```

```
        if(checksum==computedChecksum){
          Serial.printf("Checksum Verified!\n");
        }else{
          Serial.printf("Checksum Invalid!\n");
        }
      }else{
        Serial.printf("Failed to convert bits to bytes array\n");
      }

   }//if(error==false){

    // delay for 1 second to avoid repetitions
    delay(1000);
    received = false;
    syncIndex1 = 0;
    syncIndex2 = 0;

    // re-enable interrupt
    attachInterrupt(digitalPinToInterrupt(DATAPIN), handler, CHANGE);

  }//if (received == true) {

}

// ----------------------------------------------------------------------------------------
// computeTemperature
// ----------------------------------------------------------------------------------------
// 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).
// ----------------------------------------------------------------------------------------
float computeTemperature(byte highbyte, byte lowbyte){

    bool negative=false;
    Serial.printf("highbyte : 0x%02x \n",highbyte);
    if((highbyte & 0x08) == 0x08){
      negative=true;
      Serial.println("Negative");
    }

    int rawTemperature = ((highbyte & 0x0f) << 8) + lowbyte;
    Serial.printf("rawTemperature: %d\n",rawTemperature);

    if(negative){
      rawTemperature = rawTemperature - 4096;
      //limit to -40C like the acurite receiver
      if(rawTemperature < -400){
        rawTemperature = -400;
      }
    }

    float temperature = rawTemperature / 10.0;
    return temperature;
}

// ----------------------------------------------------------------------------------------
// processSignal
// ----------------------------------------------------------------------------------------
// Message is composed of:
// - Rolling Code(2 nibbles - 8 bits),
// - Status(1 nibble - 4 bits),
// - Temperature (signed binary - 3 nibbles - 12 bits ),
// - Hash Code (2 nibles - 8 bits)
//
// Each bit is composed of two signals - a seperator signal followed by a short/long
// ----------------------------------------------------------------------------------------
bool processSignal(unsigned int *bits){

    bool error = false;

    int totalSignalsInMessage = 32 *2;
```

```
    int count =0;
    for(unsigned int i=syncIndex1; i!=(syncIndex1+totalSignalsInMessage)%RING_BUFFER_SIZE; i=(i+2)%
RING_BUFFER_SIZE) {

      unsigned long t0 = timings[i];
      unsigned long t1 = timings[(i+1)%RING_BUFFER_SIZE];

      if (t0>(SEP_LENGTH-150) && t0<(SEP_LENGTH+150)){
        if (t1>(BIT1_LENGTH-1000) && t1<(BIT1_LENGTH+1000)) {
          bits[count]=1;
        } else if (t1>(BIT0_LENGTH-1000) && t1<(BIT0_LENGTH+1000)) {
          bits[count]=0;
        } else {
          bits[count]=-1;
        }
      }else {
//        Serial.printf("** t0: %d, t1: %d\n",t0,t1);
//        Serial.printf("Read Error\n");
        error=true;
        break;
      }
      count++;
    }//for

   if(error==false){

      //print bits received
      Serial.print("bits: ");
      for(int i=0;i<32;i++){
        if(bits[i]>=0){
          Serial.print(bits[i]);
        }else{
          Serial.print("X");
        }
        if((i+1)%4==0){
          Serial.printf(" ");
        }
      }
      Serial.println("");
   }
  return error;
}
// --------------------------------------------------------------------------------------
// isSync
// --------------------------------------------------------------------------------------
// Detect if a sync signal is present. A sync signal is composed of a seperator signal
// (short off 0.5ms) followed by a sync signal (9ms off).
// --------------------------------------------------------------------------------------
bool isSync(unsigned int idx) {

  //get the last two indexes from the ring buffer
  int t0Index = (idx+RING_BUFFER_SIZE-1) % RING_BUFFER_SIZE;
  unsigned long t0 = timings[t0Index];
  unsigned long t1 = timings[idx];

  //check that t0 is a seperator signal, should be around 0.5ms
  if (t0>(SEP_LENGTH-100) && t0<(SEP_LENGTH+100) ){

    //check that t1 is a sync signal, allow for anything between 8-10ms
    if (t1>(SYNC_LENGTH-1000) && t1<(SYNC_LENGTH+1000)){
      if (digitalRead(DATAPIN) == HIGH) {

//        Serial.printf("toIndex=%d, idx=%d, RING_BUFFER_SIZE=%d \n",t0Index, idx, RING_BUFFER_SIZE);
//        Serial.printf("t0=%d, t1=%d\n",t0,t1);

        //found
        return true;
      }
    }
  }
  return false;
```

```
}

// ----------------------------------------------------------------------------------------
// handler
// ----------------------------------------------------------------------------------------
// Interrupt handler - reads incoming signal changes from receiver
// ----------------------------------------------------------------------------------------
void ICACHE_RAM_ATTR handler() {
  static unsigned long duration = 0;
  static unsigned long lastTime = 0;
  static unsigned int ringIndex = 0;
  static unsigned int syncCount = 0;

  // exit out if we are in processing a signal.
  if (received == true) {
//    Serial.println("received==true");
    return;
  }

  // calculating timing since last change
  long time = micros();
  duration = time - lastTime;
  lastTime = time;

  // store data in ring buffer
  ringIndex = (ringIndex + 1) % RING_BUFFER_SIZE;
  timings[ringIndex] = duration;

  // detect sync signal
  if (isSync(ringIndex)) {

    syncCount ++;

    if (syncCount == 1) {

       // first time sync is seen, record buffer index
      syncIndex1 = (ringIndex+1) % RING_BUFFER_SIZE;

    }else if (syncCount == 2) {
      syncCount = 0;

      // second time sync is seen, start bit conversion
      syncIndex2 = (ringIndex+1) % RING_BUFFER_SIZE;

      //check how many signals we have received between syncs
      unsigned int changeCount  = 0;
      if(syncIndex2 < syncIndex1){
        changeCount = syncIndex2+RING_BUFFER_SIZE - syncIndex1;
      }else{
        changeCount = syncIndex2 - syncIndex1;
      }

      if (changeCount < 66 || changeCount > 68) {
        //too many signals, disregard
        received = false;
        syncIndex1 = 0;
        syncIndex2 = 0;
      } else {

        //we found our signal
        received = true;

      }
    }

  }
}



// ----------------------------------------------------------------------------------------
```

```
// calculateLSFR
// ------------------------------------------------------------------------------------------
void calculateLSFR() {
    int i;
    uint8_t reg = 0x7C;
    uint8_t temp_reg = 0;

    for (i = 0; i < MESSAGE_SIZE; i++) {
        temp_reg = reg & 0x01;
        reg >>= 1;
        reg |= (temp_reg << 7);

        if (temp_reg) {
            reg ^= 0x18;
        }

        LSFR_sequence[i] = reg;
        //printf("%02x\n", LSFR_sequence[i]);
    }
}
// ------------------------------------------------------------------------------------------
// combineLSFR
// ------------------------------------------------------------------------------------------
uint8_t combineLSFR(uint8_t len, uint8_t *data) {
    uint8_t hash_reg = 0; // not 0x64
    int byte_idx, bit_idx;
    uint8_t byte, bit;
    //printf("***COMBINE\n");

    for (byte_idx = 0; byte_idx < len; byte_idx++) {
        for (bit_idx = 7; bit_idx >= 0; bit_idx--) {
            bit = (data[byte_idx] & (1 << bit_idx)) >> bit_idx;
            if (bit) {
                hash_reg ^= LSFR_sequence[byte_idx * 8 + (7 - bit_idx) + LSFR_OFFSET];
                //printf("[%d]: %02x\n", byte_idx * 8 + (7 - bit_idx), hash_reg);
            }
            bit = 0;
        }
    }

    return hash_reg;
}

// ------------------------------------------------------------------------------------------
// computeChecksum
// ------------------------------------------------------------------------------------------
uint8_t computeChecksum(int length, uint8_t *buff) {
    calculateLSFR();
    return combineLSFR(length, buff);
}

// ------------------------------------------------------------------------------------------
// bitsToByteArray
// ------------------------------------------------------------------------------------------
// Convert array of integers containing the bits received into byte array
// ------------------------------------------------------------------------------------------
bool bitsToByteArray(unsigned int *bits, int len, byte* buffer, int bufferLen){

  int bidx=0;
  unsigned int sum = 0;

  for(int i=0;i<len;i++){


    //break out if we have filled up the buffer
    if(bidx >= bufferLen){
      return false;
    }

    sum += bits[i];
```

```
    //new byte?
    if((i+1)%8==0){

      //save to buffer, reset sum and increment bidx
      buffer[bidx]=sum;
      sum=0;
      bidx++;

    }else{
      sum<<=1;
    }
  }

  return true;
}
```

Presentation

References

Reference	URL
RF Protocol	http://www.osengr.org/WxShield/Downloads/Weather-Sensor-RF-Protocols.pdf
Reverse Engineer Wireless Temperature / Humidity / Rain Sensors — Part 1	https://rayshobby.net/wordpress/reverse-engineer-wireless-temperature-humidity-rain-sensors-part-1/
Acurite Receiver Code	http://raysfiles.com/arduino/temperature_display.ino
FCC Info	https://fccid.io/RNE606TXA1
On Off Keying	https://www.youtube.com/watch?v=w6V9NyXwohI