```
/* Convert RF signal into bits (rain gauge version)
 * Written by : Ray Wang (Rayshobby LLC)
 * http://rayshobby.net/?p=9056
 *
 * Customized for debugging, education, and the particular gauge, by Max Mikles -
The Mikles Projects
 * www.themiklesprojects.com
 */

// ring buffer size has to be large enough to fit
// data between two successive sync signals
#define RING_BUFFER_SIZE  256
#define DATAPIN  3  // D3 is interrupt 1

#define SYNC_HI  570
#define SYNC_LO  650

// bits 1 and 0 signals are about 610-612um duration
//  when including the HI and LO components of the wave.
#define BIT1_HI  340
#define BIT1_LO  270
#define BIT0_HI  160
#define BIT0_LO  450

#define SIG_TOL  75   //100 is more forgiving

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;

// detect if a sync signal is present
bool isSync(unsigned int idx) {

  // check if we've received 4 squarewaves of matching timing
  unsigned long t1 = 0;
  unsigned long t0 = 0;
  for(int i = 0; i < 8; i += 2) {
    t1 = timings[(idx + RING_BUFFER_SIZE - i ) % RING_BUFFER_SIZE];
    t0 = timings[(idx + RING_BUFFER_SIZE - i - 1) % RING_BUFFER_SIZE];
    if(t0 < (SYNC_HI - SIG_TOL) || t0 > (SYNC_HI + SIG_TOL) ||
       t1 < (SYNC_LO - SIG_TOL)  || t1 > (SYNC_LO + SIG_TOL)) {
      return false;
    }
  }
  //Serial.println("In sync");
```

```
  return true;
}

/* Interrupt 1 handler */
void handler() {
  static unsigned long duration = 0;
  static unsigned long lastTime = 0;
  static unsigned int ringIndex = 0;
  static unsigned int syncCount = 0;

  // ignore if we haven't processed the previous received signal
  if (received == true) {
    return;
  }
  // calculating timing since last change
  // micros() is reliable for at least one call within an interrupt handler
  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 ++;
    // first time sync is seen, record buffer index
    if (syncCount == 1) {
      syncIndex1 = (ringIndex+1) % RING_BUFFER_SIZE;
    }
    else if (syncCount == 2) {
      // second time sync is seen, start bit conversion
      syncCount = 0;
      syncIndex2 = (ringIndex+1) % RING_BUFFER_SIZE;
      unsigned int changeCount = (syncIndex2 < syncIndex1) ?
(syncIndex2+RING_BUFFER_SIZE - syncIndex1) : (syncIndex2 - syncIndex1);

      //char buffer[50] = "";
     //sprintf(buffer,"Count %d; Pointer @ %d", changeCount, syncIndex1);
     //Serial.println(buffer);

      // changeCount must be 122 -- (56 bits * 2) for data + (5 bits * 2) for sync
signal
      if (changeCount != 122 || syncIndex1 > 134) {
        received = false;
```

```
      syncIndex1 = 0;
      syncIndex2 = 0;
    }
    else {
      received = true;
    }
  }
 }
}

int t2b(unsigned int t0, unsigned int t1) {
 if (t0 > (BIT1_HI - SIG_TOL) && t0 < (BIT1_HI + SIG_TOL) &&
     t1 > (BIT1_LO - SIG_TOL) && t1 < (BIT1_LO + SIG_TOL)) {
   return 1;

 } else if (t0>(BIT0_HI - SIG_TOL) && t0 < (BIT0_HI + SIG_TOL) &&
           t1>(BIT0_LO - SIG_TOL) && t1<(BIT0_LO + SIG_TOL)){
   return 0;

 }

 // Report failure details - is signal beyond tolerance?
 //char buffer[32];
 //sprintf(buffer,"t0 %4d: t1 %4d", t0, t1);
 //Serial.println();
 //Serial.println(buffer);

 return -1;  // undefined
}

void setup() {
 Serial.begin(9600);
 Serial.println("Started.");
 pinMode(DATAPIN, INPUT);
 attachInterrupt(digitalPinToInterrupt(DATAPIN), handler, CHANGE);
}

void loop() {
 static long initial_value = -1;

 if (received == true) {

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

  Serial.println("Received");
```

```
// extract rain clicks
unsigned int startIndex, stopIndex;
unsigned long rain = 0;
bool fail = false;

//mjm report all bits
startIndex = syncIndex1;

// report each timing value
//for (int i = startIndex; i < startIndex+112; i+=2) {
//  Serial.println(i);
//}

// report all data formatted as bytes
int j = 0;
for (int i = startIndex; i < startIndex+112; i+=2) {
  if (j % 8 == 0) {
    Serial.print(" ");
  }
 Serial.print(t2b(timings[i], timings[i+1]));
  j++;
}
Serial.println();

// This gauge seems to use the lowest 7 bits of byte 3,4,5 only.
for(int byteidx=3;byteidx<6;byteidx++) {
  startIndex = (syncIndex1 + (byteidx*8+1)*2) % RING_BUFFER_SIZE;
  stopIndex =  (syncIndex1 + (byteidx*8+8)*2) % RING_BUFFER_SIZE;

  int bit = -1;
  char buffer[10];
  for(int i=startIndex; i != stopIndex; i = (i+2) % RING_BUFFER_SIZE) {
    bit = t2b(timings[i], timings[(i+1)%RING_BUFFER_SIZE]);
    rain = (rain<<1) + bit;
    if (bit < 0)  fail = true;
  }
}

if (fail) {
  Serial.println("Decoding error.");
} else {
  if (initial_value < 0) initial_value = rain;
  double d = rain;
  Serial.print("Rain = ");
  Serial.println(d/100);
```

```
    }

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

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