```
#include <SPI.h>
#include <RH_RF95.h>

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display = Adafruit_SSD1306();

#define Apin 9
#define Bpin 6
#define Cpin 5

#define LED 13

/* for feather32u4 */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 7

// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);


// timing
#define TRANSMIT_INTERVAL 10000      // interval between sending updates
#define DISPLAY_INTERVAL 150      // interval between updating display
#define MAX_FIX_AGE 5000   // Ignore data from GPS if older than this
unsigned long lastSend, lastDisplay, lastFix, lastRecv;
bool sending = false;

// 95% error radius at HDOP=1
#define GPS_BASE_ACCURACY 6.2  // m

#define ACCURACY_THRESHOLD 30  // m

// tinyGPS
#include <TinyGPS.h>

TinyGPS gps;

void setup() {
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  pinMode(LED, OUTPUT);

  pinMode(Apin, INPUT_PULLUP);
  pinMode(Bpin, INPUT_PULLUP);
  pinMode(Cpin, INPUT_PULLUP);

  // by default, we'll generate the high voltage from the 3.3v line internally! (neat!)
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C (for the 128x32)
  say("hello.", "", "", "");
  delay(3000);
  display.clearDisplay();

  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);

  while (!rf95.init()) {
    say("LoRa radio init failed", "", "", "");
    while (1);
  }

  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
  if (!rf95.setFrequency(RF95_FREQ)) {
    say("setFrequency failed", "", "", "");
    while (1);
  }

  // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on

  // The default transmitter power is 13dBm, using PA_BOOST.
  // If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
  // you can set transmitter powers from 5 to 23 dBm:
  rf95.setTxPower(23, false);

  Serial.begin(9600);
  Serial1.begin(9600);
}

#define MAGIC_NUMBER_LEN 2
uint8_t MAGIC_NUMBER[MAGIC_NUMBER_LEN] = {0x02, 0xcb};

//String timeStr = "";
uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
int lastRSSI;

// lat/lon are stored as signed 32-bit ints as millionths of a degree (-123.45678 => -123,456,780)
int32_t myLat;
int32_t myLon;
float myElev;  // unused
float myHAcc;
bool amIAccurate;
int32_t theirLat;
int32_t theirLon;
float theirElev;  // unused
bool areTheyAccurate;

void processRecv() {
  for (int i = 0; i < MAGIC_NUMBER_LEN; i++) {
    if (MAGIC_NUMBER[i] != buf[i]) {
      return;
    }
  }
  void* p = buf + MAGIC_NUMBER_LEN;
  theirLat = *(int32_t*)p;
  p = (int32_t*)p + 1;
  theirLon = *(int32_t*)p;
  p = (int32_t*)p + 1;
  areTheyAccurate = *(uint8_t*)p;
  lastRecv = millis();
}

void transmitData() {
  long sinceLastFix = millis() - lastFix;
  if (sinceLastFix > MAX_FIX_AGE) {
    // GPS data is stale
    return;
  }

  uint8_t len = 2 * sizeof(int32_t) + sizeof(uint8_t) + MAGIC_NUMBER_LEN + 1;
  uint8_t radiopacket[len];
  for (int i = 0; i < MAGIC_NUMBER_LEN; i++) {
    radiopacket[i] = MAGIC_NUMBER[i];
  }
  void* p = radiopacket + MAGIC_NUMBER_LEN;
  *(int32_t*)p = myLat;
  p = (int32_t*)p + 1;
  *(int32_t*)p = myLon;
  p = (int32_t*)p + 1;
  *(uint8_t*)p = amIAccurate;
  radiopacket[len - 1] = '\0';

  sending = true;
  rf95.send((uint8_t *)radiopacket, len);
  rf95.waitPacketSent();
  sending = false;
  lastSend = millis();
}

void loop() {
  if (Serial1.available()) {
    char c = Serial1.read();
    //Serial.write(c);
    if (gps.encode(c)) { // Did a new valid sentence come in?
      attemptUpdateFix();
    }
  }

  if (rf95.available()) {
    uint8_t len = sizeof(buf);
    if (rf95.recv(buf, &len)) {
      lastRSSI = rf95.lastRssi();
      digitalWrite(LED, HIGH);
      digitalWrite(LED, LOW);
      processRecv();
    }
  }

  long sinceLastTransmit = millis() - lastSend;
  if (sinceLastTransmit < 0 || sinceLastTransmit > TRANSMIT_INTERVAL) {
    transmitData();
  }

  long sinceLastDisplayUpdate = millis() - lastDisplay;
  if (sinceLastDisplayUpdate < 0 || sinceLastDisplayUpdate > DISPLAY_INTERVAL) {
    updateDisplay();
  }

}

void attemptUpdateFix() {
  //setFixTime();
  setFix();
}

String fixAge() {
  long elapsed = (millis() - lastRecv) / 1000;
  int n;
  char unit;
  if (elapsed < 2) {
    return "now";
  } else if (elapsed < 60) {
    n = elapsed;
    unit = 's';
  } else if (elapsed < 3600) {
    n = elapsed / 60;
    unit = 'm';
  } else {
    n = elapsed / 3600;
    unit = 'h';
  }
  return String(n) + String(unit) + " ago";
}

void updateDisplay() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println(fmtPlayaStr(theirLat, theirLon, areTheyAccurate));
  display.println(fixAge());
  display.println();
  display.println(fmtPlayaStr(myLat, myLon, amIAccurate));
  display.setCursor(60, 8);
  display.println(String(lastRSSI) + "db");

  String fixStatus = "";
  long sinceLastFix = millis() - lastFix;
  long sinceLastSend = millis() - lastSend;
  if (sinceLastFix > MAX_FIX_AGE) {
    // GPS data is stale
    fixStatus = "!";
  } else if (sending || (sinceLastSend >= 0 && sinceLastSend < 400)) {
    fixStatus = ".";
  }
  display.setCursor(120, 24);
  display.println(fixStatus);

  display.display();

  lastDisplay = millis();
}

void say(String s, String t, String u, String v) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println(s);
  display.println(t);
  display.println(u);
  display.println(v);
  display.display();
}

// production - burning man
/*
#define MAN_LAT 40786400
#define MAN_LON -119206500
#define PLAYA_ELEV 1190.  // m
#define SCALE 1.
*/

// production - afrikaburn
#define MAN_LAT -32327403
#define MAN_LON 19745329
#define PLAYA_ELEV 320.  // m
#define SCALE 1.

// testing
```

```
/*
  #define MAN_LAT 40779625
  #define MAN_LON -73965394
  #define PLAYA_ELEV 0.  // m
  #define SCALE 6.
*/

void setFix () {
  /*
    TESTING MODE
    myLat = MAN_LAT - 200*(1e-3*millis());
    myLon = MAN_LON - 200*(1e-3*millis());
    amIAccurate = true;
    lastFix = millis();
    return;
  */

  int32_t lat, lon;
  unsigned long age;
  gps.get_position(&lat, &lon, &age);
  if (age == TinyGPS::GPS_INVALID_AGE) {
    return;
  }
  lastFix = millis() - age;

  if (lat == TinyGPS::GPS_INVALID_ANGLE || lon == TinyGPS::GPS_INVALID_ANGLE) {
    lat = 0;
    lon = 0;
  }
//  Serial.println(String(flat, 6) + " " + String(flon, 6));
  myLat = lat;
  myLon = lon;

  if (gps.hdop() == TinyGPS::GPS_INVALID_HDOP) {
    myHAcc = -1;
  } else {
    myHAcc = 1e-2 * gps.hdop() * GPS_BASE_ACCURACY;
  }
  amIAccurate = (myHAcc > 0 && myHAcc <= ACCURACY_THRESHOLD);
}

String fmtPlayaStr(int32_t lat, int32_t lon, bool accurate) {
  if (lat == 0 && lon == 0) {
    return "404 cosmos not found";
  } else {
    return playaStr(lat, lon, accurate);
  }
}

//void setFixTime() {
//  int year;
//  byte month, day, hour, minute, second, hundredths;
//  unsigned long age;
//  gps.crack_datetime(&year, &month, &day, &hour, &minute, &second, &hundredths, &age);
//  timeStr =  String(hour) + ":" + String(minute) + ":" +  String(second) + "/" + String(age) +
"ms";
//}

///// PLAYA COORDINATES CODE /////

#define DEG_PER_RAD (180. / 3.1415926535)
#define CLOCK_MINUTES (12 * 60)
#define METERS_PER_DEGREE (40030230. / 360.)
// Direction of north in clock units
//#define NORTH 10.5  // hours
```

```
//#define NUM_RINGS 13  // Esplanade through L
#define ESPLANADE_RADIUS (2500 * .3048)  // m
#define FIRST_BLOCK_DEPTH (440 * .3048)  // m
#define BLOCK_DEPTH (240 * .3048)  // m
// How far in from Esplanade to show distance relative to Esplanade rather than the man
#define ESPLANADE_INNER_BUFFER (250 * .3048)  // m
// Radial size on either side of 12 w/ no city streets
#define RADIAL_GAP 2.  // hours
// How far radially from edge of city to show distance relative to city streets
#define RADIAL_BUFFER .25  // hours

//// overrides for afrikaburn
#define NORTH 3.3333  // make 6ish approx line up with bearing 80 deg
#define NUM_RINGS 0  // only give distance relative to clan


// 0=man, 1=espl, 2=A, 3=B, ...
float ringRadius(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return ESPLANADE_RADIUS;
  } else if (n == 2) {
    return ESPLANADE_RADIUS + FIRST_BLOCK_DEPTH;
  } else {
    return ESPLANADE_RADIUS + FIRST_BLOCK_DEPTH + (n - 2) * BLOCK_DEPTH;
  }
}

// Distance inward from ring 'n' to show distance relative to n vs. n-1
float ringInnerBuffer(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return ESPLANADE_INNER_BUFFER;
  } else if (n == 2) {
    return .5 * FIRST_BLOCK_DEPTH;
  } else {
    return .5 * BLOCK_DEPTH;
  }
}

int getReferenceRing(float dist) {
  for (int n = NUM_RINGS; n > 0; n--) {
    Serial.println(n + ":" + String(ringRadius(n)) + " " + String(ringInnerBuffer(n)));
    if (ringRadius(n) - ringInnerBuffer(n) <= dist) {

      return n;
    }
  }
  return 0;
}

String getRefDisp(int n) {
  if (n == 0) {
    return ")(";
  } else if (n == 1) {
    return "Espl";
  } else {
    return String(char(int('A') + n - 2));
  }
}

String playaStr(int32_t lat, int32_t lon, bool accurate) {
  // Safe conversion to float w/o precision loss.
  float dlat = 1e-6 * (lat - MAN_LAT);
  float dlon = 1e-6 * (lon - MAN_LON);

  float m_dx = dlon * METERS_PER_DEGREE * cos(1e-6 * MAN_LAT / DEG_PER_RAD);
  float m_dy = dlat * METERS_PER_DEGREE;

  float dist = SCALE * sqrt(m_dx * m_dx + m_dy * m_dy);
  float bearing = DEG_PER_RAD * atan2(m_dx, m_dy);

  float clock_hours = (bearing / 360. * 12. + NORTH);
  int clock_minutes = (int)(clock_hours * 60 + .5);
  // Force into the range [0, CLOCK_MINUTES)
  clock_minutes = ((clock_minutes % CLOCK_MINUTES) + CLOCK_MINUTES) % CLOCK_MINUTES;

  int hour = clock_minutes / 60;
  int minute = clock_minutes % 60;
  String clock_disp = String(hour) + ":" + (minute < 10 ? "0" : "") + String(minute);

  int refRing;
  if (6 - abs(clock_minutes/60. - 6) < RADIAL_GAP - RADIAL_BUFFER) {
    refRing = 0;
  } else {
    refRing = getReferenceRing(dist);
  }
  float refDelta = dist - ringRadius(refRing);
  long refDeltaRounded = (long)(refDelta + .5);

  return clock_disp + " & " + getRefDisp(refRing) + (refDeltaRounded >= 0 ? "+" : "-") +
String(refDeltaRounded < 0 ? -refDeltaRounded : refDeltaRounded) + "m" + (accurate ? "" : "-ish");
}
```