```
#include <SPI.h>
#include<nRF24L01.h>
#include <RF24.h>

#include<FastLED.h>

#define NUM_LEDS 8 //LED
#define DATA_PIN 6 //LED

#define CE_PIN   7  //WIFI
#define CSN_PIN 8 //WIFI


const int motorPin = 5; //motor

CRGB leds[NUM_LEDS];
RF24 radio(CE_PIN, CSN_PIN); // CE, CSN  WIFI
const byte address[6] = "00001"; //WIFI
const int buttonBeginPin = 2;
const int buttonSnoozePin = 3;
int buttonBeginState = 0;
int buttonSnoozeState = 0;

char dataToSnooze[] = "snooze";
char dataToSend[] = "singal comeIn";
char dataToStart[] = "count start";
char dataToBegin[] = "counters Begin";
char dataToReStart[] = "counter Restart";
char dataToReBegin[] = "counters ReBegin";


char txNum = '0';
```

```
int ackData[2] = {-1, -1}; // to hold the two
values coming from the slave
char repData[32];
char result[32];

bool timesUp = false;
bool restTime = false;
bool rslt = true;
bool start = false;
bool begins = false;
bool snooze = false;
bool timeAfterRest = false;

unsigned long snoozeStart;
unsigned long singalComein;
unsigned long RestTimeLeft;

unsigned long coolDownTime = 30000;

unsigned long currentMillis;
unsigned long prevMillis;
unsigned long txIntervalMillis = 500; // send once
per second

void setup()
{

 Serial.begin(9600);
 Serial.println("Sender Starting");

 radio.begin();
 radio.setDataRate( RF24_250KBPS );
```

```
 radio.enableAckPayload();
  radio.setRetries(5,5); // delay, count
                                        // 5 gives a
1500 µsec delay which is needed for a 32 byte
ackPayload
 radio.openWritingPipe(address);

 FastLED.addLeds<WS2812, DATA_PIN>(leds,
NUM_LEDS);  // GRB ordering is assumed LED灯条

 pinMode(motorPin,OUTPUT); //motor
 pinMode(buttonBeginPin, INPUT);
 pinMode(buttonSnoozePin, INPUT);



}

void loop() {
 buttonBeginState = digitalRead(buttonBeginPin);
 buttonSnoozeState = digitalRead(buttonSnoozePin);



  delay(100);
  currentMillis = millis();
  if (currentMillis - prevMillis >=
txIntervalMillis) {
      send();
      timesUps();

     if(buttonBeginState == HIGH){
        begins = true;
         }
```

```
     Serial.print("  buttonBeginState: ");
     Serial.println(buttonBeginState);
     Serial.print("  buttonSnoozeState: ");
     Serial.println(buttonSnoozeState);
     Serial.print("  restTime: ");
     Serial.println(restTime);
     Serial.print("  restTimeLeft: ");
     Serial.println(RestTimeLeft);

     if(receiverOutOfRange(singalComein) &&
timesUp == false && restTime == false && snooze ==
false){
        restTime = true;
        RestTimeLeft = singalComein;
         }
     if((restTime == true) && (millis() -
RestTimeLeft < coolDownTime) ){
        leds[0] = CRGB::Yellow;
        leds[1] = CRGB::Yellow;
        leds[2] = CRGB::Yellow;
        leds[3] = CRGB::Yellow;
        leds[4] = CRGB::Yellow;
        leds[5] = CRGB::Yellow;
        leds[6] = CRGB::Yellow;
        leds[7] = CRGB::Yellow;
        FastLED.show();
        digitalWrite(motorPin,LOW);
        radio.write( &dataToReStart,
sizeof(dataToReStart) );
        radio.read(&repData, sizeof(repData));
        strcpy(result, repData);
        if(strcmp(result, "Times up Again") == 0){
```

```
          timesUp = true;
          RestTimeLeft = millis();
           }
        timesUps();
         }
     if((restTime == true) && (millis() -
RestTimeLeft >= coolDownTime) ){
        leds[0] = CRGB::Red;
        leds[1] = CRGB::Red;
        leds[2] = CRGB::Red;
        leds[3] = CRGB::Red;
        leds[4] = CRGB::Red;
        leds[5] = CRGB::Red;
        leds[6] = CRGB::Red;
        leds[7] = CRGB::Red;
        FastLED.show();
       digitalWrite(motorPin,LOW);
        restTime = false;
        timeAfterRest = true;
        singalComein = millis();
       radio.read(&repData, sizeof(repData));
        start = true;
        }

  }


}
void send() {


   rslt = radio.write( &dataToSend,
```

```
sizeof(dataToSend) );
       // Always use sizeof() as it gives the size
as the number of bytes.
       // For example if dataToSend was an int
sizeof() would correctly return 2
    Serial.print("  rslt: ");
   Serial.println(rslt);
    Serial.print("  Data Sent: ");
   Serial.println(dataToSend);

    if (rslt && restTime == false) {
      if(start){
       radio.read(&repData, sizeof(repData));
       strcpy(result, repData);

       for(int i = 0; i < 100 ; i++){
         if(start && strcmp(result, "counter
Rebegin") == 0){
           start = false;
           }
          radio.write( &dataToReBegin,
sizeof(dataToReBegin) );
          }
          }

     if(begins){
       for(int i = 0; i < 100 ; i++){
         radio.write( &dataToStart,
sizeof(dataToStart) );
          }
         begins = false;
       }
```

```
     if ( radio.isAckPayloadAvailable() ) {
       singalComein = millis();
        radio.read(&repData, sizeof(repData));
        strcpy(result, repData);
        if(strcmp(result, "Times up") == 0){
          timesUp = true;
           }
        Serial.print("  Data receive: ");
        Serial.println(result);
     }
     else {
        Serial.println("  Acknowledge but no data
");
     }

   }
   else {
     Serial.print("  singalComein :  ");
    Serial.println(singalComein);
     Serial.print(" OutOfRange : ");
      Serial.
println(receiverOutOfRange(singalComein));
    if(receiverOutOfRange(singalComein)){
       timesUp = false;
     }
     Serial.println("  receive failed  ");
   }

  prevMillis = millis();
 }
```

```
void showData() {

  Serial.print("  Acknowledge data ");
 Serial.print(ackData[0]);
  Serial.print(", ");
 Serial.println(ackData[1]);
  Serial.println();

}

void timesUps(){
    if(timesUp == true){
      if(buttonSnoozeState == HIGH){
        snoozeStart = millis();
        snooze = true;
        timesUp = false;

         }
      // Turn the LED on, then pause
      leds[0] = CRGB::Green;
      leds[1] = CRGB::Green;
      leds[2] = CRGB::Green;
      leds[3] = CRGB::Green;
      leds[4] = CRGB::Green;
      leds[5] = CRGB::Green;
      leds[6] = CRGB::Green;
      leds[7] = CRGB::Green;
      FastLED.show();
     digitalWrite(motorPin, HIGH);
      delay(1000);
      // Now turn the LED off, then pause
      leds[0] = CRGB::White;
```

```
      leds[1] = CRGB::White;
      leds[2] = CRGB::White;
      leds[3] = CRGB::White;
      leds[4] = CRGB::White;
      leds[5] = CRGB::White;
      leds[6] = CRGB::White;
      leds[7] = CRGB::White;
      FastLED.show();
     digitalWrite(motorPin,LOW);
      delay(2000);
    }
    if(snooze == true && timesUp == false &&
restTime == false ){
      ifsnooze();
     snoozeCounter(snoozeStart);
       }
    if(timesUp == false && restTime == false){
    leds[0] = CRGB::Blue;
    leds[1] = CRGB::Green;
    leds[2] = CRGB::Red;
    leds[3] = CRGB::White;
    leds[4] = CRGB::Blue;
    leds[5] = CRGB::Green;
    leds[6] = CRGB::Red;
    leds[7] = CRGB::White;
    FastLED.show();
   digitalWrite(motorPin,LOW);
      }
}
void ifsnooze(){
    leds[0] = CRGB::Yellow;
    leds[1] = CRGB::Yellow;
```

```
    leds[2] = CRGB::Yellow;
    leds[3] = CRGB::Yellow;
    leds[4] = CRGB::Yellow;
    leds[5] = CRGB::Yellow;
    leds[6] = CRGB::Yellow;
    leds[7] = CRGB::Yellow;
    FastLED.show();
   digitalWrite(motorPin, LOW);
    delay(1500);
    // Now turn the LED off, then pause
    leds[0] = CRGB::Black;
    leds[1] = CRGB::Black;
    leds[2] = CRGB::Black;
    leds[3] = CRGB::Black;
    leds[4] = CRGB::Black;
    leds[5] = CRGB::Black;
    leds[6] = CRGB::Black;
    leds[7] = CRGB::Black;
    FastLED.show();
   digitalWrite(motorPin,LOW);
    delay(1500);
    radio.write( &dataToSnooze,
sizeof(dataToSnooze) );
}
bool receiverOutOfRange(unsigned long singalComein){
  if(start == false && (millis() - singalComein >=
5000)){
    return true;
    }else{
      return false;
       }
}
```

```
void snoozeCounter(unsigned long snoozeStart){
  if(snooze == true && (millis() - snoozeStart >=
50000)){
    snooze = false;
    timesUp = true;
   radio.write( &dataToReStart,
sizeof(dataToReStart) );
    }
}
```