Introduction to Microcontrollers

Experiment 3:

(KEY FOR ECE AND CYENG STUDENTS)

Communication with LoRa Packet Radios

Developed 2019 by S. Rowland and M. Eckels

Contents

1 Objectives

To learn to utilize LoRa wireless radio modules for server and client communication to transmit the state of a PIR (Pyroelectric InfraRed) sensor.

2 Background

2.1 What is a PIR sensor?

PIR sensors allow you to sense motion, almost always used to detect whether a human has moved in or out of the sensors range. They are small, inexpensive, low-power, easy to use and don't wear out. For that reason, they are commonly found in appliances and gadgets used in homes or businesses. They are often referred to as PIR, "Passive Infrared", "Pyroelectric", or "IR motion" sensors.

2.1.1 How a PIR sensor works

PIRs are basically made of a pyroelectric sensor (which you can see below as the round metal can with a rectangular crystal in the center), which can detect levels of infrared radiation. Everything emits some low level radiation, and the hotter something is, the more radiation is emitted. The sensor in a motion detector is actually split in two halves. The reason for that is that we are looking to detect motion (change) not average IR levels. The two halves are wired up so that they cancel each other out. If one half sees more or less IR radiation than the other, the output will swing high or low.

Along with the pyroelectric sensor is a bunch of supporting circuitry, resistors and capacitors. It seems that most small hobbyist sensors use the BISS0001 ("Micro Power PIR Motion Detector IC"), undoubtedly a very inexpensive chip. This chip takes the output of the sensor and does some minor processing on it to emit a digital output pulse from the analog sensor.

2.2 What is a LoRa radio module?

The LoRa Radio Module is a type of long range low data rate data radio modem based on Sx1276 from Semtech. It is a low-cost sub-1 GHz transceiver module designed for operations in the unlicensed ISM (Industrial Scientific Medical) and LPRD bands. Frequency spectrum modulation/demodulation, multi-channel operation, high bandwidth efficiency and anti-blocking performance make LoRa modules easy to realize the robust and reliable wireless link.

Also, these packet radios are simpler than WiFi or BLE, you don’t have to associate, pair, scan, or worry about connections. All you do is send data whenever you like, and any other modules tuned to that same frequency (and, with the same encryption key) will receive. The receiver can then send a reply back. The modules do packetization, error correction and can also auto-retransmit so it’s not like you have worry about everything, but less power is wasted on maintaining a link or pairing.

2.2.1 Long Range Wireless Radio

LoRa, essentially, is a clever way to get very good receiver sensitivity and low bit error rate (BER) from inexpensive chips. That means low-data rate applications can get much longer range using LoRa rather than using other comparably priced radio technologies.

2.2.2 LoRaWAN

LoRaWAN is different. It is a media access control (MAC)-layer protocol built on top of LoRa, built using Semtech's LoRa modulation scheme. LoRaWAN, however, is rarely used for industrial (private network) applications. It is a better fit for public wide-area networks because all the channels are tuned to the same frequencies; for single-area use, it's better to have only one network operating in order to avoid collision problems.

LoRaWAN will not be used in this lab, but it follows the same protocol and ease-of-use.

2.3 Arduino Open Source Libraries

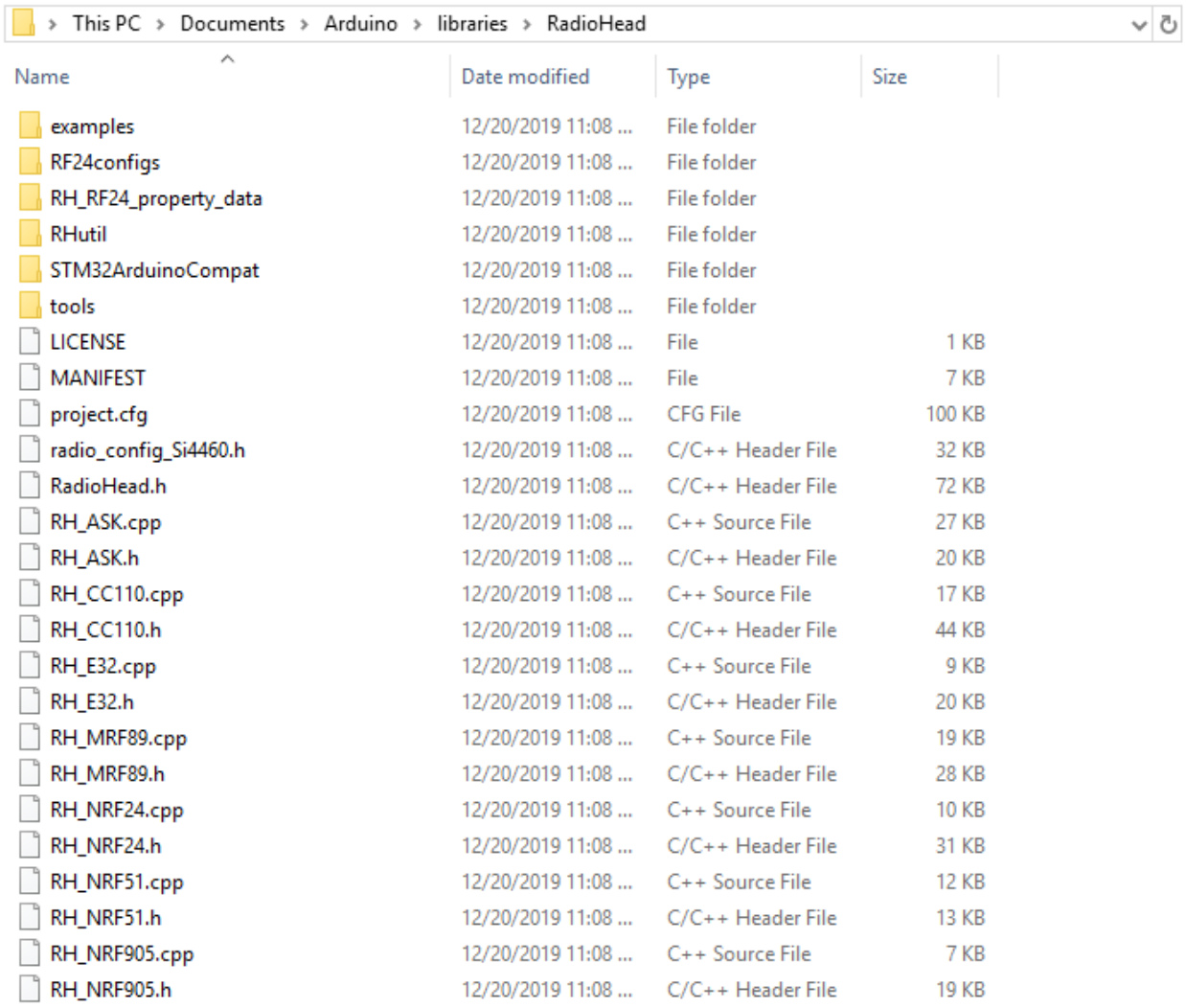
2.3.1 Arduino Libraries

Similar to the previous lab, we will need to install secondary Arduino libraries to allow the communication and handling of the LoRa module. The library that we will be using – RadioHead – is not included in a global repository to allow downloading from the library manager within the Arduino IDE. We will need to install it separately.

2.3.2 Installing Open Source Libraries

To install an external library that is not included in a repository, we will need to start with finding and downloading the library. The RadioHead library can be found at github.com/adafruit/RadioHead.

On GitHub, click the Clone or download drop down and download the Zip file. Extract the zip file to the libraries folder under the Arduino folder in the Documents folder in your user account (C:\Users\xxxxxxxx\Documents\Arduino\libraries\RadioHead). Verify that the RadioHead folder at this location contains something similar to the following:



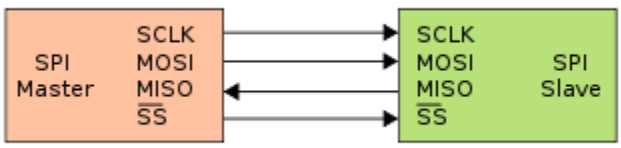
Name	Date modified	Type	Size
examples	12/20/2019 11:08 ...	File folder	
RF24configs	12/20/2019 11:08 ...	File folder	
RH_RF24_property_data	12/20/2019 11:08 ...	File folder	
RHutil	12/20/2019 11:08 ...	File folder	
STM32ArduinoCompat	12/20/2019 11:08 ...	File folder	
tools	12/20/2019 11:08 ...	File folder	
LICENSE	12/20/2019 11:08 ...	File	1 KB
MANIFEST	12/20/2019 11:08 ...	File	7 KB
project.cfg	12/20/2019 11:08 ...	CFG File	100 KB
radio_config_Si4460.h	12/20/2019 11:08 ...	C/C++ Header File	32 KB
RadioHead.h	12/20/2019 11:08 ...	C/C++ Header File	72 KB
RH_ASK.cpp	12/20/2019 11:08 ...	C++ Source File	27 KB
RH_ASK.h	12/20/2019 11:08 ...	C/C++ Header File	20 KB
RH_CC110.cpp	12/20/2019 11:08 ...	C++ Source File	17 KB
RH_CC110.h	12/20/2019 11:08 ...	C/C++ Header File	44 KB
RH_E32.cpp	12/20/2019 11:08 ...	C++ Source File	9 KB
RH_E32.h	12/20/2019 11:08 ...	C/C++ Header File	20 KB
RH_MRF89.cpp	12/20/2019 11:08 ...	C++ Source File	19 KB
RH_MRF89.h	12/20/2019 11:08 ...	C/C++ Header File	28 KB
RH_NRF24.cpp	12/20/2019 11:08 ...	C++ Source File	10 KB
RH_NRF24.h	12/20/2019 11:08 ...	C/C++ Header File	31 KB
RH_NRF51.cpp	12/20/2019 11:08 ...	C++ Source File	12 KB
RH_NRF51.h	12/20/2019 11:08 ...	C/C++ Header File	13 KB
RH_NRF905.cpp	12/20/2019 11:08 ...	C++ Source File	7 KB
RH_NRF905.h	12/20/2019 11:08 ...	C/C++ Header File	19 KB

2.4 SPI Communication

The LoRa radio modules will communicate with the Arduino with the SPI protocol as opposed to the previous lab which utilized the I²C protocol.

2.4.1 SPI Protocol

SPI devices communicate in full duplex mode using a master-slave architecture (alternate terminology being main and secondary) with a single master. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS), sometimes called chip select (CS), lines.



The SPI bus specifies four logic signals:

- SCLK: Serial Clock (output from master)
- MOSI: Master Output Slave Input, or Master Out Slave In (data output from master)
- MISO: Master Input Slave Output, or Master In Slave Out (data output from slave)
- SS: Slave Select (often active low, output from master)

2.4.2 Advantages of SPI Communication

- Full duplex communication in the default version of this protocol
- Push-pull drivers (as opposed to open drain) provide good signal integrity and high speed
- Higher throughput than I²C or SMBus. Not limited to any maximum clock speed, enabling potentially high speed
- Complete protocol flexibility for the bits transferred
- Not limited to 8-bit words
- Arbitrary choice of message size, content, and purpose
- Extremely simple hardware interfacing
- Typically, lower power requirements than I²C or SMBus due to less circuitry (including pull up resistors)
- Slaves use the master's clock and do not need precision oscillators
- Slaves do not need a unique address – unlike I²C or GPIB or SCSI
- Uses only four pins on IC packages, and wires in board layouts or connectors, much fewer than parallel interfaces
- At most one unique bus signal per device (chip select); all others are shared
- Signals are unidirectional allowing for easy galvanic isolation
- Simple software implementation

2.4.3 Disadvantages of SPI Communication

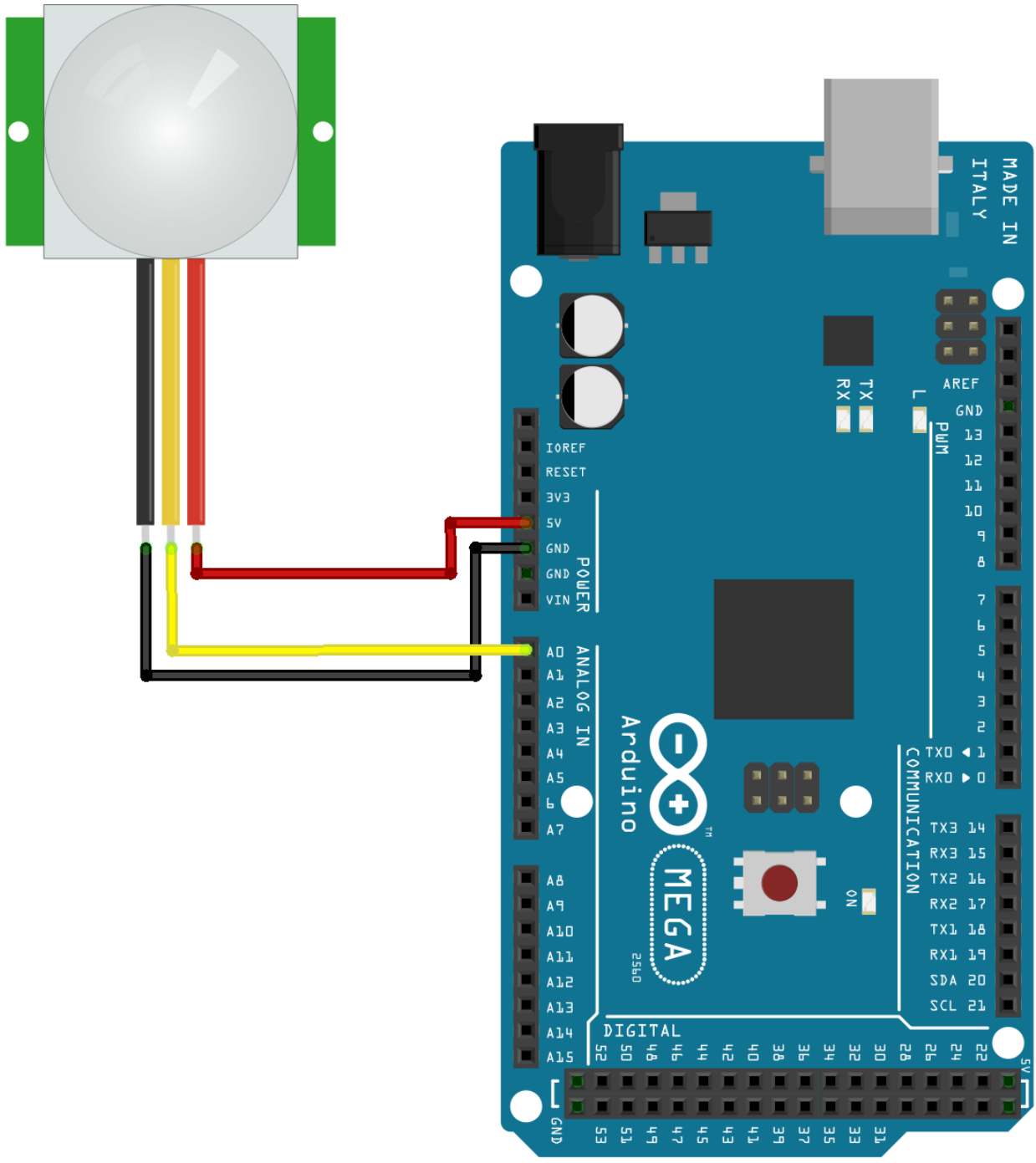
- Requires more pins on IC packages than I²C, even in the three-wire variant
- No in-band addressing; out-of-band chip select signals are required on shared buses
- No hardware flow control by the slave (but the master can delay the next clock edge to slow the transfer rate)
- No hardware slave acknowledgment (the master could be transmitting to nowhere and not know it)
- Typically supports only one master device (depends on device's hardware implementation)
- No error-checking protocol is defined
- Without a formal standard, validating conformance is not possible
- SPI does not support hot swapping (dynamically adding nodes).

3 Procedures

3.1 Reading from the PIR Sensor

3.1.1 Wiring the PIR Sensor

Looking at the PIR Sensor with the plastic dome on top and the 3 pins below closest to you, from left-to-right the wiring is GND, OUTPUT, VCC. To verify this, take off the plastic dome and observe the printed text on the PCB.



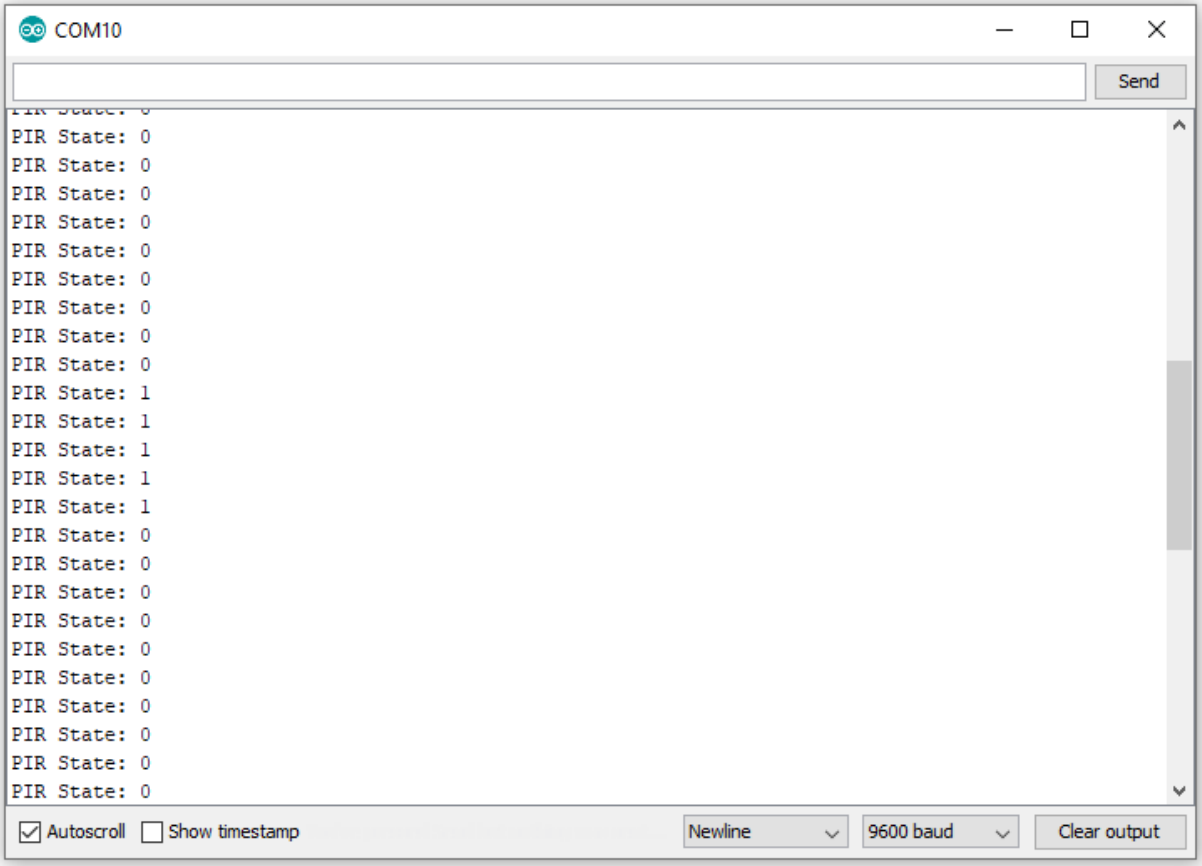
3.1.2 Programming the Arduino

The output pin of the PIR sensor gives a digital output corresponding to a trigger event of an object warmer than the ambient air temperature moving in front of the dome. With this in mind, the digital pin can be checked for a HIGH trigger to determine if the PIR was tripped.

```
#define PIRPIN A0

void setup() {
  pinMode(PIRPIN, INPUT);

  Serial.begin(9600);
}

void loop() {
  bool pirState = digitalRead(PIRPIN);

  Serial.print("PIR State: ");
  Serial.println(pirState);

  delay(500);
}
```

3.1.3 Testing the PIR Sensor

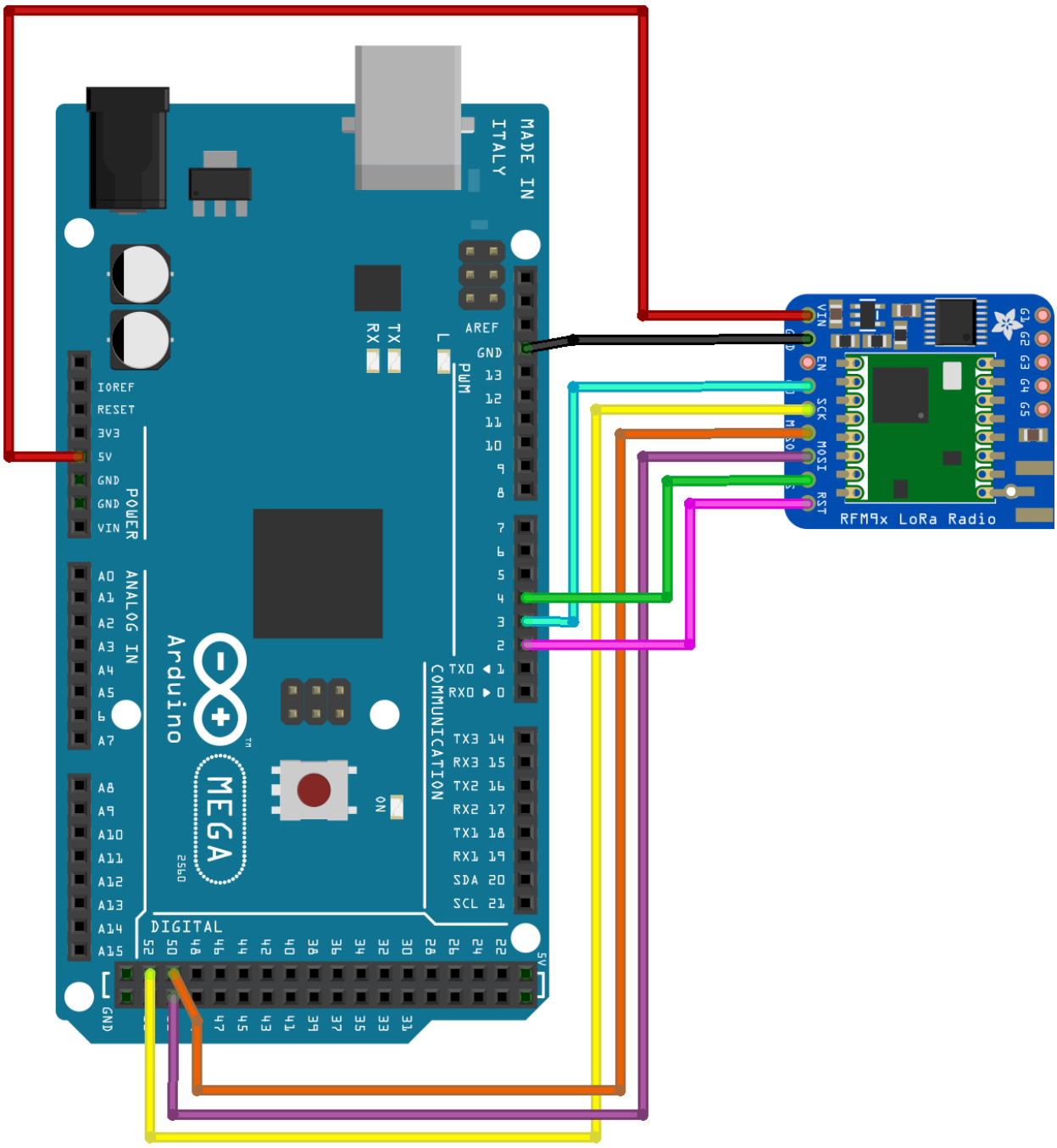
Once the PIR is wired to the Arduino and the Arduino is programmed with the code above, it can be tested by watching the output, waving your hand in front of the PIR, and then verifying that there are a few lines in the serial monitor where the PIR state reports with a value of 1.



3.2 Setting up the LoRa Server

3.2.1 Wiring the LoRa Module

Wire the LoRa module to the Arduino Mega as follows:



3.2.2 Programming the Arduino

The following example code gives what you need to send a message to a client and listen for a response, modify this code with the code from section 3.1 to make a server that asks a client for the state of the PIR sensor. Coordinate with the class to guarantee that your definition of MY_ADDRESS does not overlap with other students.

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

```
#define RFM95_CS 4
#define RFM95_RST 2
#define RFM95_INT 3

#define RF95_FREQ 915.0

// who am i? (server address)
#define MY_ADDRESS     2

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

// Class to manage message delivery and receipt, using the driver declared
above
RHReliableDatagram rf95_manager(rf95, MY_ADDRESS);

#define LED 13

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

  while (!Serial);
  Serial.begin(9600);
  delay(100);

  Serial.println("Arduino LoRa RX Test!");

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

  while (!rf95_manager.init()) {
    Serial.println("LoRa radio init failed");
    while (1);
  }
  Serial.println("LoRa radio init OK!");

  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while (1);
  }
  Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

  rf95.setTxPower(13, false);
}

uint8_t data[] = "And hello back to you";
```

```
uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];

void loop()
{
  if (rf95_manager.available())
  {
    // Wait for a message addressed to us from the client
    uint8_t len = sizeof(buf);
    uint8_t from;
    if (rf95_manager.recvfromAck(buf, &len, &from)) {
      buf[len] = 0; // zero out remaining string

      Serial.print("Got packet from #"); Serial.print(from);
      Serial.print(" [RSSI :");
      Serial.print(rf95.lastRssi());
      Serial.print("] : ");
      Serial.println((char*)buf);

      // Send a reply back to the originator client
      if (!rf95_manager.sendtoWait(data, sizeof(data), from))
        Serial.println("Sending failed (no ack)");
    }
  }
}
```

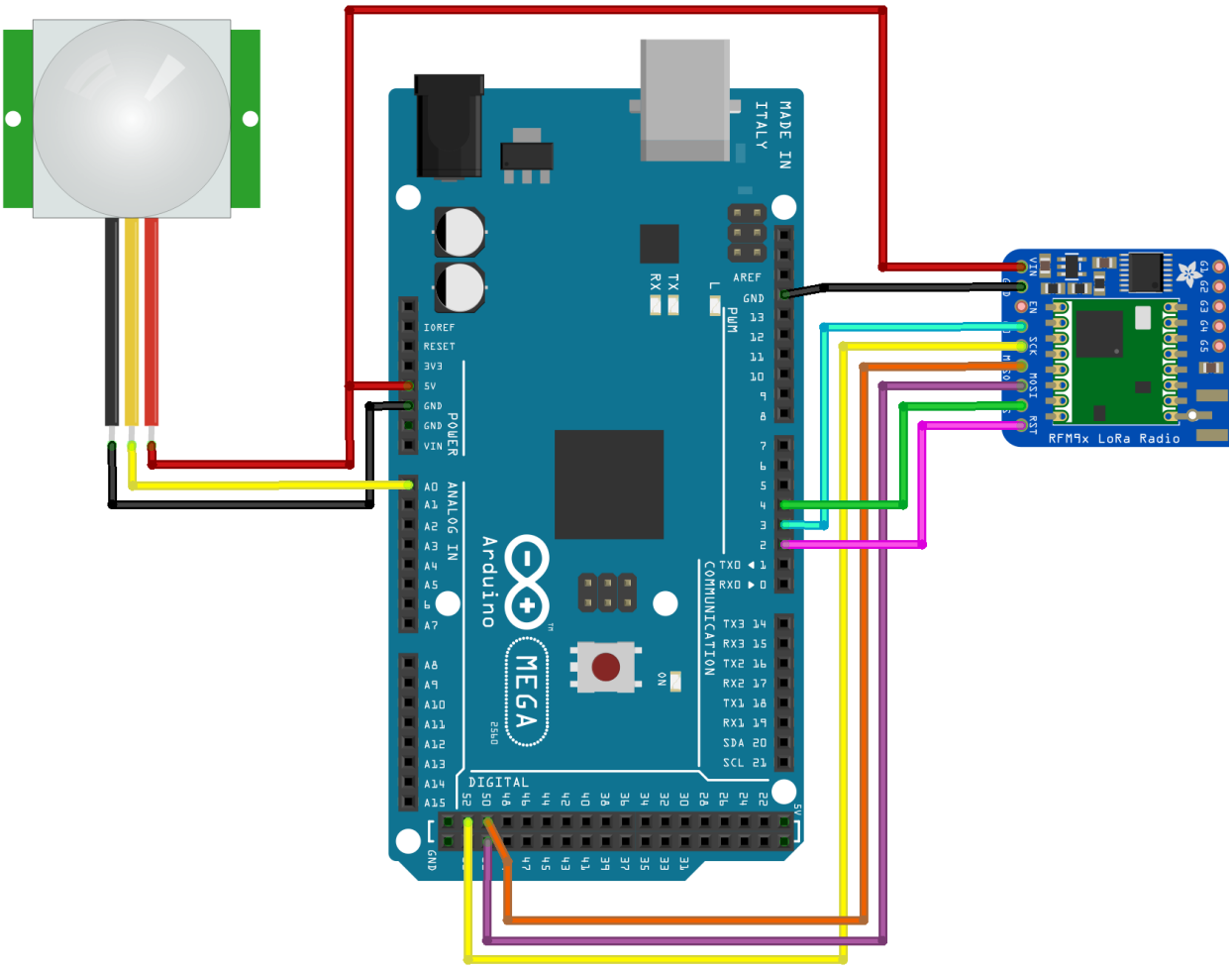
3.2.3 Starting the Server

To start the server, upload the code that was modified to the Arduino and connect to the serial port – make sure the terminal is set to the baud rate of 9600.

3.3 Setting up the LoRa Client

3.3.1 Wiring the LoRa Module and PIR Sensor

Wire the LoRa module and PIR sensor to the Arduino Mega as follows:



3.3.2 Programming the Arduino

The following example code gives what you need to receive a message from a server and then send a response, modify this code with the code from section 3.1 to make a server that asks a client for the state of the PIR sensor. Coordinate with the class to guarantee that your definition of MY_ADDRESS and DEST_ADDRESS does not overlap with other students. The DEST_ADDRESS (destination address) must be the same as your MY_ADDRESS definition in section 3.2.

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

#define RFM95_CS 4
#define RFM95_RST 2
#define RFM95_INT 3

#define RF95_FREQ 915.0

#define MY_ADDRESS     1
// Where to send packets to!
#define DEST_ADDRESS   2
```

```
RH_RF95 rf95(RFM95_CS, RFM95_INT);
RHReliableDatagram rf95_manager(rf95, MY_ADDRESS);

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

  while (!Serial);
  Serial.begin(9600);
  delay(100);

  Serial.println("Arduino LoRa TX Test!");

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

  while (!rf95_manager.init()) {
    Serial.println("LoRa radio init failed");
    while (1);
  }
  Serial.println("LoRa radio init OK!");

  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while (1);
  }
  Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

  rf95.setTxPower(13, false);
}

uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
uint8_t data[] = "  OK";
int16_t packetnum = 0;  // packet counter, we increment per xmission

void loop()
{
  char radiopacket[20] = "Hello World #      ";
  itoa(packetnum++, radiopacket+13, 10);
  Serial.print("Sending "); Serial.println(radiopacket);

  // Send a message to the DESTINATION!
  if (rf95_manager.sendtoWait((uint8_t *)radiopacket, strlen(radiopacket),
DEST_ADDRESS)) {
    // Now wait for a reply from the server
    uint8_t len = sizeof(buf);
    uint8_t from;
    if (rf95_manager.recvfromAckTimeout(buf, &len, 2000, &from)) {
```

```
      buf[len] = 0; // zero out remaining string

      Serial.print("Got reply from #"); Serial.print(from);
      Serial.print(" [RSSI :");
      Serial.print(rf95.lastRssi());
      Serial.print("] : ");
      Serial.println((char*)buf);

    } else {
      Serial.println("No reply, is anyone listening?");
    }
  } else {
    Serial.println("Sending failed (no ack)");
  }

  delay(1000);  // Wait 1 second between transmits, could also 'sleep' here!
}
```

3.3.3 Connecting the Client to the Server

To start the client, upload the code that was modified to the Arduino and connect to the serial port – make sure the terminal is set to a baud rate of 9600.

4 Study Questions & Deliverables

1. Provide a comprehensive report that demonstrates your completion of this laboratory assignment. Key sections to include in your report are "**Introduction and Background**", "**Methodologies**", and "**Results and Conclusions**" with inclusion of figures, tables, codes, and so forth in different sections.
2. What are the key benefits to LoRa modules over XBee modules?
3. What are the downsides to LoRa modules over XBee modules?
4. What applications could this type of radio communication and server/client infrastructure serve?
5. Which devices that you have used might be using a technology similar to the LoRa wireless modules?

Instructor: Dr. Shayan (Sean) Taheri.

Note – Cheating and Plagiarism: Cheating and plagiarism are not permitted in any form and cause certain penalties. The instructor reserves the right to fail culprits.

Deliverable: All your responses to the assignment questions should be included in a single compressed file to be uploaded in the Gannon University (GU) – Blackboard Learn environment.

5 Equipment

Name	Quantity
Arduino Mega Microcontroller	2
USB-A to USB-B Cable	2
Male-to-Female Dupont Jumpers	12
Breadboard	1 or 2
LoRa Radio Modules	2
PIR Sensor	1

LoRa Radio Modules

1. LoRa Radio Module - 868MHz - DFRobot

2. Simple Arduino LoRa Communciation (8km) - Arduino Project Hub

3. The Arduino Guide to LoRa® and LoRaWAN® | Arduino Documentation | Arduino Documentation

4. How to use LoRa with Arduino | LoRa Tutorial with Circuit Code and AT commands | Reyax RYLR896 - YouTube

5. How to use LoRa with Arduino | LoRa Tutorial with Circuit Code and AT commands | Reyax RYLR896 - YouTube