**LORA COMMUNICATION PROTOCOL FOR FARMING**

A Dissertation submitted in partial fulfillment of requirement for the award of the

Degree of

**BACHELOR OF TECHNOLOGY**

In

**ELECTRONICS AND INSTRUMENTATION ENGINEERING**

BY

**G. KARTHIK (12BD1A1010)**

**MOHAMMED SALAHUDDIN (12BD1A1034)**

**G. SAI CHARAN (12BD1A1050)**

**Y. KRISHNA SHASHANK (12BD1A1057)**

**Under the esteemed guidance of**

**Mr. T. VIKRAM**

Asst. Professor, E.C.E Dept.

****

**Department of Electronics and Instrumentation Engineering**

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

Approved by A.I.C.T.E

(Affiliated to JNT University, Hyderabad)

3-5-1026 Narayanaguda, Hyderabad

Telangana-500029

2014-2015

**ACKNOWLEDGEMENT**

With great pleasure we want to take this opportunity to express our heartful gratitude to all the people who have helped us in making this project work a grand success.

First of all, we would like to thank God for giving me courage and strength to finish this work. Also,we would like to thank **Mr.VINAY PATANKAR Project Guide, HYDERABAD** for his continued guidance and support during the course of this thesis along with our **Internal Guide Mrs. Swapna** for her constant guidance throughout our project work. We would like to thank her for her patience and time to guide us at every step of this work.

I take this opportunity to thank all who have rendered their full support to my project work.

I also render my thanks to **DR. MAHESWAR DUTTA, Principal** who encouraged us in doing this project. I grateful to **MR.NEIL GOGTE, Director** for falicitating all aminities required for the successful completition of this project.

We would also like to thank our **HOD, Mr.VINAY PATANKAR, EIE Department,** for his guidance and support. We have no words to describe the support offered to us by our lecturers and other faculty.

# Chapter 1 Wireless Communications

## 1.1 Overview of Wireless Communications

Wireless communications is, by any measure, the fastest growing segment of the communications industry. As such, it has captured the attention of the media and the imagination of the public. Cellular systems have experienced exponential growth over the last decade and there are currently around two billion users worldwide. Indeed, cellular phones have become a critical business tool and part of everyday life in most developed countries, and are rapidly supplanting antiquated wire line systems in many developing countries. In addition, wireless local area networks currently supplement or replace wired networks in many homes, businesses, and campuses. Many new applications, including wireless sensor networks, automated highways and factories, smart homes and appliances, and remote telemedicine, are emerging from research ideas to concrete systems. The explosive growth of wire-less systems coupled with the proliferation of laptop and palmtop computers indicate a bright future for wireless networks, both as stand-alone systems and as part of the larger networking infrastructure. However, many technical challenges remain in designing robust wireless networks that deliver the performance necessary to support emerging applications.

## 1.2 Current Wireless Systems

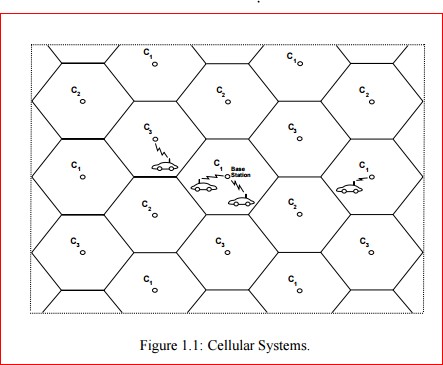
### 1.2.1 Cellular Telephone Systems

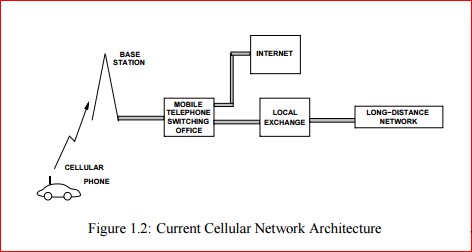
A cellular network or mobile network is a communication network where the last link is wireless. The network is distributed over land areas called cells, each served by at least one fixed-location [transceiver,](https://en.wikipedia.org/wiki/Transceiver) known as a [cell site](https://en.wikipedia.org/wiki/Cell_site) or [base station.](https://en.wikipedia.org/wiki/Base_station) This base station provides the cell with the network coverage which can be used for transmission of voice, data and others. In a cellular network, each cell uses a different set of frequencies from neighboring cells, to avoid interference and provide guaranteed bandwidth within each cell.

When joined together these cells provide radio coverage over a wide geographic area. This enables a large number of portable transceivers (e.g., [mobile phones,](https://en.wikipedia.org/wiki/Mobile_phone) [pagers,](https://en.wikipedia.org/wiki/Pager) etc.) to communicate with each other and with fixed transceivers and telephones anywhere in the network, via base stations, even if some of the transceivers are moving through more than one cell during transmission.

Cellular networks offer a number of desirable features:

* More capacity than a single large transmitter, since the same frequency can be used for multiple links as long as they are in different cells
* Mobile devices use less power than with a single transmitter or satellite since the cell towers are closer
* Larger coverage area than a single terrestrial transmitter, since additional cell towers can be added indefinitely and are not limited by the horizon





### 1.2.2 Cordless Phones

A cordless telephone or portable telephone replaces the handset cord with a [radio](https://en.wikipedia.org/wiki/Radio) link. The handset communicates with a [base station](https://en.wikipedia.org/wiki/Base_station) connected to a fixed [telephone line.](https://en.wikipedia.org/wiki/Telephone_line) The range is limited, usually to the same building or some short distance from the base station. The base station attaches to the telephone network the same way a corded telephone does.

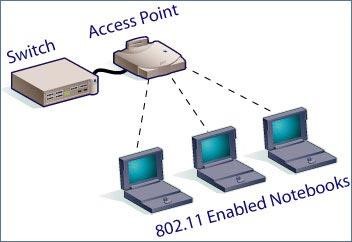
The base station on subscriber premises is what differentiates a cordless telephone from a [mobile telephone.](https://en.wikipedia.org/wiki/Mobile_telephones) Current cordless telephone standards, such as [PHS](https://en.wikipedia.org/wiki/Personal_Handy-phone_System) and [DECT,](https://en.wikipedia.org/wiki/DECT) have blurred the once clear-cut line between cordless and mobile telephones by implementing cell [handoff](https://en.wikipedia.org/wiki/Handoff) (handover); various advanced features, such as data-transfer; and even, on a limited scale, international [roaming.](https://en.wikipedia.org/wiki/Roaming) In specialized models, base stations are maintained by a commercial [mobile network operator](https://en.wikipedia.org/wiki/Mobile_network_operator) and users subscribe to the service.



**Figure 1.3 Cordless phone**

### 1.2.3 Wireless LANs

Wireless LANs provide high-speed data within a small region, e.g. a campus or small building, as users move from place to place. Wireless devices that access these LANs are typically stationary or moving at pedestrian speeds. All wireless LAN standards in the U.S. operate in unlicensed frequency bands. The primary unlicensed bands are the ISM bands at 900 MHz, 2.4 GHz, and 5.8 GHz, and the Unlicensed National Information Infrastructure (U-NII) band at 5 GHz. In the ISM bands, unlicensed users are secondary users so must cope with interference from primary users when such users are active. There are no primary users in the UNII band. An FCC license is not required to operate in either the ISM or U-NII bands. However, this advantage is a double-edged sword, since other unlicensed systems operate in these bands for the same reason, which can cause a great deal of interference between systems. The interference problem is mitigated by setting a limit on the power per unit bandwidth for unlicensed systems. Wireless LANs can have either a star architecture, with wireless access points or hubs placed throughout the coverage region, or a peer-to-peer architecture, where the wireless terminals self-configure into a network.

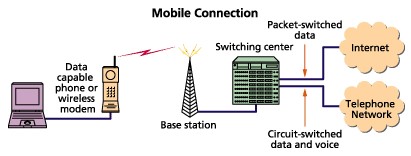


**Figure 1.4 Wireless LAN**

### 1.2.4 Wide Area Wireless Data Services

Wide area wireless data services provide wireless data to high-mobility users over a very large coverage area. In these systems a given geographical region is serviced by base stations mounted on towers, rooftops, or mountains. The base stations can be connected to a backbone wired network or form a multihop ad hoc wireless network.

Initial wide area wireless data services had very low data rates, below 10 Kbps, which gradually increased to 20 Kbps. There were two main players providing this service: Motient and Bell South Mobile Data (formerly RAM Mobile Data). Metricom provided a similar service with a network architecture consisting of a large network of small inexpensive base stations with small coverage areas. The increased efficiency of the small coverage areas allowed for higher data rates in Metricom, 76 Kbps, than in the other wide-area wireless data systems. However, the high infrastructure cost for Metricom eventually forced it into bankruptcy, and the system was shut down. Some of the infrastructure was bought and is operating in a few areas as Ricochet.

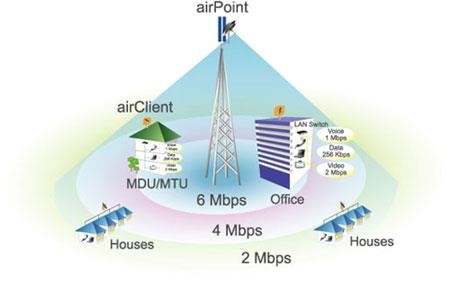


**Figure 1.5 Wide Area Network**

### 1.2.5 Broadband Wireless Access

Broadband wireless access provides high-rate wireless communications between a fixed access

point and multiple terminals. These systems were initially proposed to support interactive video service to the home, but the application emphasis then shifted to providing high speed data access (tens of Mbps) to the Internet, the WWW, and to high speed data networks for both homes and businesses. In the U.S. two frequency bands were set aside for these systems: part of the 28 GHz spectrum for local distribution systems (local multipoint distribution systems or LMDS) and a band in the 2 GHz spectrum for metropolitan distribution systems (multichannel multipoint distribution services or MMDS). LMDS represents a quick means for new service providers to enter the already stiff competition among wireless and wireline broadband service providers. MMDS is a television and telecommunication delivery system with transmission ranges of 30-50 Km. MMDS has the capability to deliver over one hundred digital video TV channels along with telephony and access to emerging interactive services such as the Internet. MMDS will mainly compete with existing cable and satellite systems. Europe is developing a standard similar to MMDS called Hiperaccess.



**Figure 1.6 Broadbamd Wireless Access**

### 1.2.6 Paging Systems

Paging systems broadcast a short paging message simultaneously from many tall base stations or satellites trans-mitting at very high power (hundreds of watts to kilowatts). Systems with terrestrial transmitters are typically localized to a particular geographic area, such as a city or metropolitan region, while geosynchronous satellite transmitters provide national or international coverage. In both types of systems no location management or routing functions are needed, since the paging message is broadcast over the entire coverage area. The high complexity and power of the paging transmitters allows lowcomplexity, low-power, pocket paging receivers with a long usage time from small and lightweight batteries. In addition, the high transmit power allows paging signals to easily penetrate building walls. Paging service also costs less than cellular service, both for the initial device and for the monthly usage charge, although this price advantage has declined considerably in recent years as cellular prices dropped. The low cost, small and lightweight handsets, long battery life, and ability of paging devices to work almost anywhere indoors or outdoors are the main reasons for their appeal.

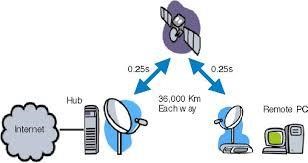


**Figure 1.7 Paging System**

### 1.2.7 Satellite Networks

Commercial satellite systems are another major component of the wireless communications infrastructure. Geosynchronous systems include Inmarsat and OmniTRACS. The former is geared mainly for analog voice transmission from remote locations. For example, it is commonly used by journalists to provide live reporting from war zones. The first generation Inmarsat-A system was designed for large (1m parabolic dish antenna) and rather expensive terminals. Newer generations of Inmarsats use digital techniques to enable smaller, less expensive terminals, around the size of a briefcase. Qualcomm’s OmniTRACS provides two-way communications as well as location positioning.

The system is used primarily for alphanumeric messaging and location tracking of trucking fleets.

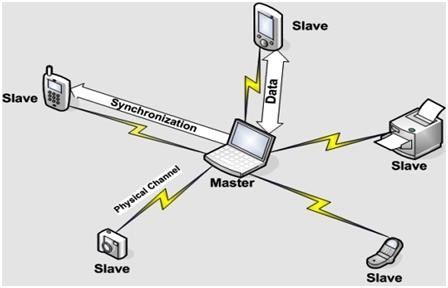


**Figure 1.8 Satellite Network**

### 1.2.8 Low-Cost Low-Power Radios: Bluetooth and Zigbee

As radios decrease their cost and power consumption, it becomes feasible to embed them in more types of electronic devices, which can be used to create smart homes, sensor networks, and other compelling applications. Two radios have emerged to support this trend: Bluetooth and Zigbee.

Bluetoothradios provide short range connections between wireless devices along with rudimentary networking capabilities. The Bluetooth standard is based on a tiny microchip incorporating a radio transceiver that is built into digital devices. The transceiver takes the place of a connecting cable for devices such as cell phones, laptop and palmtop computers, portable printers and projectors, and network access points. Bluetooth is mainly for short range communications, e.g. from a laptop to a nearby printer or from a cell phone to a wireless headset.



#### Figure 1.9 Bluetooth Network

The ZigBee3 radio specification is designed for lower cost and power consumption than Bluetooth.

The specification is based on the IEEE 802.15.4 standard. The radio operates in the same ISM band as

Bluetooth, and is capable of connecting 255 devices per network. The specification supports data rates of up to 250 Kbps at a range of up to 30 m. These data rates are slower than Bluetooth, but in exchange the radio consumes significantly less power with a larger transmission range. The goal of ZigBee is to provide radio operation for months or years without recharging, thereby targeting applications such as sensor networks and inventory tags.



**Figure 1.10 Zigbee**

### 1.2.9 Ultrawideband Radios

Ultra-wideband (UWB) radios are extremely wideband radios with very high potential data rates.

The concept of ultra-wideband communications actually originated with Marconi’s spark gap transmitter, which occupied a very wide bandwidth. However, since only a single low-rate user could occupy the spectrum, wideband communications was abandoned in favor of more efficient communication techniques. The renewed interest in wideband communications was spurred by the

FCC’s decision in 2002 to allow operation of UWB devices as system underlayed beneath existing users over a 7 GHz range of frequencies. These systems can operate either at baseband or at a carrier frequency in the 3.6-10.1 GHz range. The underlay in theory interferers with all systems in that frequency range, including critical safety and military systems, unlicensed systems such as 802.11 wireless and Bluetooth, and cellular systems where operators paid billions of dollars for dedicated spectrum use. The FCC’s ruling was quite controversial given the vested interest in interference-free spectrum of these users. To minimize the impact of UWB on primary band users, the FCC put in place severe transmit power restrictions. This requires UWB devices to be within close proximity of their intended receiver.

## 1.3 Differences

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Technology** | **Range** | **Frequency**  **of**  **Operation** | **Data**  **Transfer Rate** | **Voltage** | **Operating Temperature** |
| Zigbee | (30 - 100)m | 2.4GHz | 115 KBps | 3.3 V | -40 to 85 C |
| Bluetooth | 100 m | 2.4 GHz | 25MBps | 2 V | -40 to 70 C |
| Wireless LAN | 32 m | 2.4GHz /  5GHz | 433 MBps | 3.3 V | 0 to 40 C |
| LoRa | 20 Km | 434 MHz | 0.3 to 50  KBps | 3.3 V | -40 to 85 C |

**Table 1.1 Differences Between Few Wireless Communications**

# Chapter 2 Arduino

## 2.1 Overview

The Mega 2560 is a microcontroller board based on the [ATmega2560.](http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf) It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. The Mega 2560 board is compatible with most shields designed for the Uno and the former boards Duemilanove or Diecimila.

The Mega 2560 is an update to the [Arduino Mega,](https://www.arduino.cc/en/Main/en/Main/ArduinoBoardMega) which it replaces. **2.2 Technical specs**

|  |  |
| --- | --- |
| Microcontroller | ATmega 2560 |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limit) | 6-20V |
| Digital I/O Pins | 54 (of which 15 provide PWM output) |
| Analog Input Pins | 16 |
| DC Current per I/O Pin | 20 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 256 KB of which 8 KB used by bootloader |
| SRAM | 8 KB |
| EEPROM | 4 KB |
| Clock Speed | 16 MHz |
| Length | 101.52 mm |
| Width | 53.3 mm |
| Weight | 37 g |

### Table 2.1 Technical Specifications of Arduino Mega 2560



**Figure 2.1 Arduino Board**

## 2.3 Programming

The Mega 2560 board can be programmed with the Arduino Software IDE. The ATmega2560 on the Mega 2560 comes preprogrammed with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (InCircuit Serial Programming) header using Arduino ISP or similar.

The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

* On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
* On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode. You can then use [Atmel’s FLIP software (](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3886)Windows) or the [DFU programmer (](http://dfu-programmer.sourceforge.net/)Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader).

## 2.4 Warnings

The Mega 2560 has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

## 2.5 Power

The Mega 2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

* Vin. The input voltage to the board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
* 5V. This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
* 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
* GND. Ground pins.
* IOREF. This pin on the board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

## 2.6 Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

## 2.7 Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50 k ohm. A maximum of 40mA is the value that must not be exceeded to avoid permanent damage to the microcontroller.

In addition, some pins have specialized functions:

* Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX). Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
* External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). These pins can be configured to trigger an interrupt on a low level, a rising or falling edge, or a change in level.
* PWM: 2 to 13 and 44 to 46. Provide 8-bit PWM output with the analogWrite() function.
* SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). These pins support SPI communication using theSPIlibrary. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Arduino /Genuino Uno and the old Duemilanove and Diecimila Arduino boards.
* LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
* TWI: 20 (SDA) and 21 (SCL). Support TWI communication using theWire library. Note that these pins are not in the same location as the TWI pins on the old Duemilanove or Diecimila Arduino boards.

The Mega 2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

* AREF. Reference voltage for the analog inputs. Used with analogReference().
* Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

## 2.8 Communication

The Mega 2560 board has a number of facilities for communicating with a computer, another board, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega16U2 (ATmega 8U2 on the revision 1 and revision 2 boards) on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino Software (IDE) includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2/ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega 2560's digital pins.

The Mega 2560 also supports TWI and SPI communication. The Arduino Software (IDE) includes a Wire library to simplify use of the TWI bus.

## 2.9 Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega 2560 PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega 2560 is designed to be compatible with most shields designed for the Uno and the older Diecimila or Duemilanove Arduino boards. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Furthermore, the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts

0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega 2560 and Duemilanove / Diecimila boards. Please note that I2C is not located on the same pins on the Mega 2560 board (20 and 21) as the Duemilanove / Diecimila boards (analog inputs 4 and 5).

## 2.10 Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Mega 2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino Software (IDE) uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega 2560 board is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the ATMega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega 2560 board contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line.

## 2.11 Revisions

The Mega 2560 does not use the FTDI USB-to-serial driver chip used in past designs. Instead, it features the ATmega16U2 (ATmega8U2 in the revision 1 and revision 2 Arduino boards) programmed as a USB-to-serial converter. Revision 2 of the Mega 2560 board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into [DFU mode.](https://www.arduino.cc/en/Hacking/DFUProgramming8U2)

Revision 3 of the Arduino board and the current Genuino Mega 2560 have the following improved features:

* 1.0 pinout: SDA and SCL pins - near to the AREF pin - and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the board that uses ATSAM3X8E, that operate with 3.3V. The second one is a not connected pin that is reserved for future purposes.
* Stronger RESET circuit.
* Atmega 16U2 replace the 8U2.

# Chapter 3 LoRa Modulation Basics

## 3.1 Introduction

LoRa is a proprietary spread spectrum modulation scheme that is derivative of Chirp Spread Spectrum modulation (CSS) and which trades data rate for sensitivity within a fixed channel bandwidth. It implements a variable data rate, utilizing orthogonal spreading factors, which allows the system designer to trade data rate for range or power, so as to optimize network performance in a constant bandwidth. LoRa is a PHY layer implementation and is agnostic with to higher-layer implementations. This allows LoRa to coexist and interoperate with existing network architectures. This application note explains some of the basic concepts of LoRa modulation and the advantages that this modulation scheme can provide when deploying both fixed and mobile low-power real-world communications networks.

## 3.2 Spread Spectrum Communications

### 3.2.1 Shannon – Hartley Theorem

No discussion on spread spectrum techniques would be complete without a brief recap of the Shannon – Hartley Theorem. In information theory, the Shannon–Hartley theorem states the maximum rate at which information can be transmitted over a communications channel of a specified bandwidth in the presence of noise. The theorem establishes Shannon's channel capacity for a communication link and defines the maximum data rate (information) that can be transmitted within a specified bandwidth in the presence of noise interference:

 **Equation 3.1**

Where:

C = channel capacity (bit/s)

B = channel bandwidth (Hz)

S = average received signal power (Watts)

N = average noise or interference power (Watts)

S/N = signal to noise ratio (SNR) expressed as a linear power ratio

By rearranging Equation 2.1 from log base 2 to the natural log, e, and by noting that we can manipulate the equation as follows:



#### Equation 3.2

For spread spectrum applications the signal to noise ratio is small, since the signal power is often below the noise floor. Assuming a noise level such that S/N << 1, Equation 2 can be rewritten as:



or



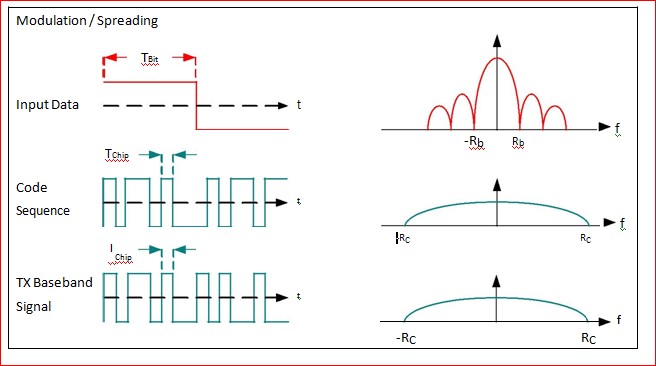
#### Equation 3.3

From equation 3 it can be seen that to transmit error free information in a channel of fixed noiseto-signal ratio, only the transmitted signal bandwidth need be increased.

### 3.2.2 Spread-Spectrum Principles

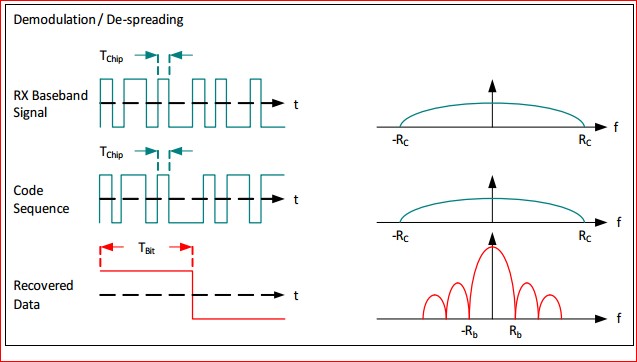
As has been noted above, by increasing the bandwidth of the signal we can compensate for the degradation of the signal-to-noise (or noise-to-signal) ratio of a radio channel.

In traditional Direct Sequence Spread Spectrum (DSSS) systems, the carrier phase of the transmitter changes in accordance with a code sequence. This process is generally achieved by multiplying the wanted data signal with a spreading code, also known as a chip sequence. The chip sequence occurs at a much faster rate than the data signal and thus spreads the signal bandwidth beyond the original bandwidth occupied by just the original signal. Note that the term chip is used to distinguish the shorter coded bits from the longer un-coded bits of the information signal.



#### Figure 3.1: Modulation / Spreading Process

At the receiver, the wanted data signal is recovered by re-multiplying with a locally generated replica of the spreading sequence. This multiplication process in the receiver effectively compresses the spread signal back to its original un-spread bandwidth, as illustrated below in Figure 2.2. It should be noted that the same chip sequence or code must be used in the receiver as in the transmitter to correctly recover the information.



#### Figure 3.2: Demodulation / De-spreading Process

The amount of spreading, for direct sequence, is dependent on the ratio of "chips per bit" - the ratio of the chip sequence to the wanted data rate, is referred to as the processing gain (Gp), commonly expressed in dB.



**Equation 3.4** Where:

RC = chip rate (Chips/second)

Rb = bit-rate (bits/second)

As well as providing inherent processing gain for the wanted transmission (which enables the receiver to correctly recover the data signal even when the SNR of the channel is a negative value); interfering signals are also reduced by the process gain of the receiver. These are spread beyond the desired information bandwidth and can be easily removed by filtering.

DSSS is widely used in data communication applications. However, challenges exist for low-cost or power-constrained devices and networks.

Often, as is the case with GPS or the DSSS PHY of IEEE Standard 802.15.4k, the system will require a highly accurate and expensive reference clock source. In addition, the longer the spreading code or sequence, the longer the time required by the receiver to perform a correlation over the entire length of the code sequence, or by either searching sequentially through code sequences or implementing multiple correlates in parallel.

This is especially of concern for power-constrained devices that cannot be “always-on” and thus need to repeatedly and rapidly synchronize.

### 3.2.3 Chirp Spread Spectrum

Chirp Spread Spectrum was developed for radar applications in the 1940’s. Traditionally used in a number of military and secure communications applications; over the past twenty years this modulation technique has seen increased adoption in a number of data communications applications due to its relatively low transmission power requirements and inherent robustness from channel degradation mechanisms such as multipath, fading, Doppler and in-band jamming interferers.

A CSS PHY was adopted by the IEEE for the Low-Rate Wireless Personal Area Networks (LRWPANs) standard 802.15.4 for applications requiring longer range and mobility than that achievable with the O-QPSK DSSS PHY mode.

## 3.3 LoRa Spread Spectrum

Semtech’s LoRa modulation addresses all of the issues associated with DSSS systems to provide a low-cost, low-power, yet above all robust alternative to the traditional spread-spectrum communications techniques.

In LoRa modulation the spreading of the spectrum is achieved by generating a chirp signal that continuously varies in frequency. An advantage of this method is that timing and frequency offsets between transmitter and receiver are equivalent, greatly reducing the complexity of the receiver design. The frequency bandwidth of this chirp is equivalent to the spectral bandwidth of the signal. The wanted data signal is chipped at a higher data rate and modulated onto the chirp signal. The relationship between the wanted data bit rate, symbol rate and chip rate for LoRa modulation can be expressed as follows:

We can define the modulation bit rate, Rb, as:



**Equation 3.5** Where:

SF = spreading factor (7.12)

BW = modulation bandwidth (Hz)

## 3.4 Key Properties of LoRa Modulation

### 3.4.1 Bandwidth Scalable

LoRa modulation is both bandwidth and frequency scalable. It can be used for both narrowband frequency hopping and wideband direct sequence applications. Unlike existing narrowband or wideband modulation schemes, LoRa can be easily adapted for either mode of operation with only a few simple configuration register changes.

### 3.4.2 Constant Envelope / Low-Power

Similar to FSK, LoRa is a constant envelope modulation scheme which means that the same lowcost and low-power high-efficiency PA stages can be re-used without modification. In addition, due to the processing gain associated with LoRa, the output power of the transmitter can be reduced compared to a conventional FSK link while maintaining the same or better link budget.

### 3.4.2 High Robustness

Due to the high BT product (BT > 1) and their asynchronous nature a LoRa signal is very resistant to both in-band and out-of-band interference mechanisms. Since the LoRa symbol period can be longer than the typical short-duration burst of fast-hopping FHSS systems, it provides for excellent immunity to pulsed AM interference mechanisms; typical receiver out-of-channel selectivity figures of 90 dB and co-channel rejection of better than 20 dB can be obtained. This compares to typically 50 dB for adjacent and alternate channel rejection, and -6 dB co-channel rejection for FSK modulation.

### 3.4.3 Multipath / fading Resistant

The chirp pulse is relatively broadband and thus LoRa offers immunity to multipath and fading, making it ideal for use in urban and suburban environments, where both mechanisms dominate.

### 3.4.4 Doppler Resistant

Doppler shift causes a small frequency shift in the LoRa pulse which introduces a relatively negligible shift in the time axis of the baseband signal. This frequency offset tolerance mitigates the requirement for tight tolerance reference clock sources. LoRa is ideal for mobile data communications links such as wireless tire-pressure monitoring systems, drive-by applications such as toll booth and mobile tag readers, and trackside communications for railroad infrastructure.

### 3.4.5 Long Range Capability

For a fixed output power and throughput, the link budget of LoRa exceeds that of conventional FSK. When taken into conjunction with the proven robustness to interference and fading mechanisms, this improvement in link budget can readily translate to x4 and beyond enhancement in range.

### 3.4.6 Enhanced Network Capacity

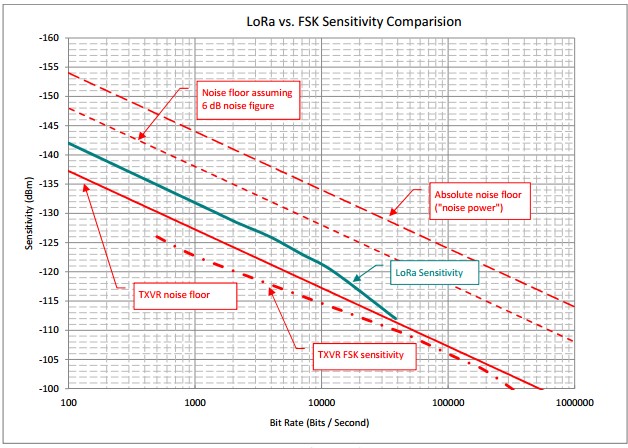
Semtech LoRa modulation employs orthogonal spreading factors which enables multiple spread signals to be transmitted at the same time and on the same channel without minimal degradation the RX sensitivity. Modulated signals at different spreading factors appear as noise to the target receiver and can be treated as such.

### 3.4.7 Ranging / Localization

An inherent property of LoRa is the ability to linearly discriminate between frequency and time errors. LoRa is the ideal modulation for radar applications and is thus ideally suited for ranging and localization applications such as real-time location services.

## 3.5 FSK vs. LoRa Sensitivity Comparison

The principle of increasing the wanted signal bandwidth to transmit error free data over longer distance (i.e. in the presence of an increasing noise-to-signal ratio) is a fundamental principal of spreadspectrum communications and can be visualized by comparing the sensitivity of LoRa vs. a competitive landscape of FSK transceivers available as illustrated in Figure 3, below:



**Figure 3.3: Comparison of LoRa and FSK Sensitivity**

The theoretical absolute noise floor (“noise power”) at room temperature assumes from Equation 1 that the channel capacity is 1 bit per Hz of bandwidth (the required SNR is thus 0 dB) and can be calculated from:



**Equation 3.6** Where:

Noise Floor=equivalent noise power(dBm)

K = Boltzmann’s Constant (~1.38 \*1023)

T=293kelvin(“roomtemperature”)

B = channel bandwidth (Hz)

1000 = scaling factor from Watt to mWatts This can be simplified as:



**Equation 3.7** Where:

-174 = 10 \* log10 (k \* T \* 1000) as defined above B = channel bandwidth (Hz) as before

TXVR noise floor indicates a close approximation to the noise floor of the current generation of subGHz FSK transceiver devices currently available and again is calculated from:



**Equation 3.8** Where:

B\*1.5 = Idealised channel bandwidth for GFSK modulation (Hz)

DSNR = Required demodulator Eb/N0 for coherent FSK (~ 10 dB)

NF = Receiver architecture noise figure (6 dB)

Compare the theoretical noise floor to typical sensitivity figures obtained from the datasheets of current generation FSK transceiver devices. It can be shown that at low data rates, specified sensitivity diverges from the theoretical RX noise floor due to the increase channel (filter) bandwidth required to compensate for expected frequency errors between the transmitter and target receiver.

Sub-GHz LoRa sensitivity is as per the specification in both the SX1272 and SX1276 datasheets and takes into account the typical 6 dB noise figure of the receiver architecture.

The Eb/N0 of LoRa offers typically 10 dB improvement over that for GFSK and thus it can be seen that LoRa offers a significant sensitivity improvement over FSK. It should be noted that if the noise figure is added to the absolute noise floor plot, the sensitivity achievable with LoRa modulation is within 6 dB of the relative noise floor.

## 3.6 Considerations for Wireless Communications

### 3.6.1 Wireless Network

#### 3.6.1.1 Star Network Topology

A star network is the most common form of network topology for power constrained end-point nodes and is relatively simple to implement. Typically a central coordinator or concentrator acts as the conduit for all network traffic. All network transmissions are routed via the central coordinator.

A star-network topology helps minimize the amount of network traffic. For a network that is not link-constrained only 3 devices and two links are involved in any communications between two nodes. In addition nodes are isolated from one another and provides for ease of replacing nodes. Centralization allows for inspection of all network traffic at a single point.

A disadvantage of this topology is that failure of the coordinator will disable all network communications.

#### 3.6.1.2 Mesh Network Topology

In a mesh network data propagates through the network via every node. Mesh networks can be considered flooding, whereby each node relays the same message regardless of the end destination or routing, whereby the method propagates along a path to its destination. Networks typically employ lookup tables or are self-routing.

Advantages of a mesh networks are the ability to “self-heal” and reconfigure themselves in the event of a loss of connectivity to a node or group of nodes.

A disadvantage of this topology is the relatively increased complexity over traditional star networks and an increase in network traffic due to the inherent in-built redundancy of the network. In addition the increased traffic that each node has to handle means that mesh networks are typically implemented in circumstances where the nodes are not power constrained.

### 3.6.2 Multipath Propagation Mechanisms

Multipath is the propagation phenomenon that results in the transmitted radio signal reaching the receiver by two or more paths. Multipath mechanisms include reflection from objects such as building, mountains, large bodies of water, atmospheric ducting, ionospheric reflection and refraction. It should be noted that these mechanisms can give rise to both constructive and destructive interference. This later case causes fading to be observed.

Multipath gives rise to small-scale fading effects:

· Rapid changes in signal strength over a small travelled distance or time interval

· Frequency drift and bandwidth spread effects caused by Doppler shifts on each multipath signal

Multipath fading mechanisms can be considered as flat or frequency-selective fading. In the case of flat fading, the bandwidth of the propagation channel is greater than that of the transmitted signal. In this case, while the spectral properties of the signal are unaltered at reception, the amplitude of the signal fluctuates with time due to changes in the gain of the channel caused by multipath. Narrow-band FSK systems attempt to mitigate for the effects of flat-fading by implementing spectral diversity techniques such as frequency hopping.

Frequency-selective fading is said to occur when the bandwidth of the propagation channel is less

than that of the transmitted signal. It can be seen that the flat-fading case is most common for narrowband FSK modulation; although for high-data rate wideband and FSK modulation inter-symbol interference can be introduced by multipath delay spreading causing distortion of the demodulated signal. To overcome frequency-selective fading, high-data rate FSK systems may implement multi-level (or m-ary) modulation to reduce the transmitted signal bandwidth. Multi-level FSK requires both more complicated receiver architecture to successfully demodulate the transmitted signal and a higher SNR than two-level FSK.

In addition to multipath fading, Doppler fading mechanisms may also need to be considered in the case of mobile communication.

As has been noted, the relatively broadband nature and high BT of LoRa provides for excellent immunity to multipath and fading mechanisms.

### 3.6.3 Link Budget

The link budget of a wireless system or network is a measure of all the gains and losses from the transmitter, through the propagation channel, to the target receiver. These gains and losses include system gains and losses associated with the antenna, matching networks, etc. as well as losses associated propagation channel itself (either though modelling or measured data).

Typically randomly varying channel mechanisms such as multipath and Doppler fading are taken into account by factoring additional margin depending on the anticipated severity.

The link budget of a network wireless link can be expressed as:



**Equation 3.9** Where:

PRX = the expected power incident at the receiver PTX

= the transmitted power

GSYSTEM = system gains such as those associated with directional antennas, etc. LSYSTEM = losses associated with the system such as feed-lines, antennas (in the case of electrical short antennas associated with many remote devices), etc.

LCHANNEL = losses due to the propagation channel, either calculated via a wide range of channel models or from empirical data

M = fading margin, again either calculated or from empirical data

A communications channel is said to be link limited when the losses associated with the channel

cause the incident power level at the receiver to be below that required to meet the SNR requirement of the receiver for correct demodulation of the received data.

### 3.6.4 Interference Limited Links

In practice, operating in license-exempt spectrum provides for no quality of service guarantee (as opposed to licensed operation, whereby the network operator has paid a fee for exclusive access to the spectrum being used) and a device operating in license-exempt spectrum is likely to be interference rather than link-budget limited.

A typical scenario may see several co-located networks attempting to access the same frequency space at the same time. While there are a number of collision mitigation mechanisms that can be implemented, either through regulation (such as LBT or transmitter duty-cycle limits) or through voluntary mechanisms such as CSMA / CSMA-CA, by the nature of dynamic interference mechanisms a channel assessment at the transmitter may not necessarily coincide with the channel conditions at the target device and the transmission may be blocked.

To avoid interference mechanisms, narrowband systems often implement frequency agility (or frequency hopping) to avoid repeated operation on the same channel or frequency. As has been noted, this frequency agility is also used by narrowband systems to mitigate for multipath propagation properties.

However, the pseudo-random nature of the hop sequence employed (as is typically required by regulation) can lead to a loss of a packet and subsequent channel synchronization due to either the transmitter or target receiver jumping to an already occupied channel or having another transmitting device hop to that frequency during a wanted transmission.

The requirement to frequency hop also leads to an increase in packet redundancy. While modern narrowband receivers typically require only a short preamble sequence for synchronization there will be a requirement to retransmit a message header so that the receiver can be assured that the received broadcast is intended. In addition, most frequency hopping systems only remain on a channel for a few milliseconds so as to minimize the probability of the channel becoming blocked by another unwanted transmission, thus in the case of a low data rate narrowband transmission the message overhead increases still further.

Any loss of synchronization between transmitter and receiver will require the devices to undergo a period of re-discovery and synchronization.

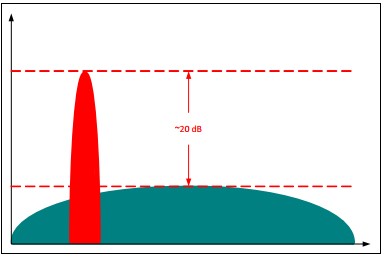
### 3.6.5 Network Coexistence

Traditionally narrowband FSK signals have traded upon their frequency agility and ability to

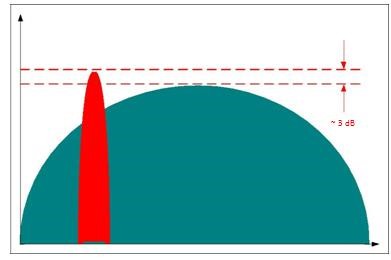
“punch through” wideband spread-spectrum signals (which are traditionally limited by power spectral density limits and thus transmit at a lower power than narrowband signals) as is illustrated below in Figure 3.4.

However, recent changes in to the measurement guidance procedures allowed by the FCC in the United States now permit wideband spread-spectrum devices to transmit at significantly higher output power levels while still complying with power spectral density limits.

Taking advantage of this latest guidance a wideband LoRa modulated signal of 500 kHz 6 dB bandwidth can transmit up to typically +27 dBm, close to the +30 dBm permitted for narrowband modulation, as illustrated in Figure 2.5, without any channel dwell time restrictions.



#### Figure 3.4: Traditional Narrowband Signal vs. Wideband Interferer



**Figure 3.5: Narrowband Signal vs. Wideband Interferer**

As has been noted, with spread spectrum modulation, the impact of interfering signals is reduced by the process gain that is inherent to the modulation. These interfering signals are spread beyond the desired information bandwidth and can be easily removed by filtering.

With narrowband modulation, interfering signals are not spread by the demodulation process. As can be seen from Figure 5, a wideband interferer will block the narrowband transmission, causing the packet to be lost.

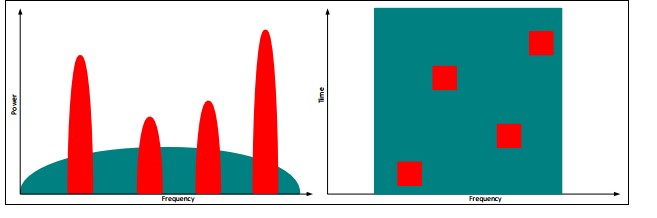
There have been many studies into the impact of dissimilar modulation and co-located networks. In Europe, 4G-LTE has been allocated frequency spectrum vacated by analog terrestrial TV and extends to 862 MHz. Studies by both OFCOM and the ECC have shown that significant interference may be expected. In addition, studies by both industry associations and notified bodies show that license-exempt radios deployed for SmartGrid applications co-located within 25 m (75’) of LTE mobile units may expect to experience significant packet loss.

In the United States, studies have also shown that the deployment of new Multilateration Location and Monitoring Service (M-LMS) services in the upper-third of the 902 – 928 MHz band could render up to 4 MHz and beyond of spectrum unsuitable for tower-based narrowband devices, such as those deployed as SmartGrid concentrators and network coordinators. Indeed, the ambient noise level

“seen” by a tower based device can have significant impact upon the realizable link budget.

In addition, the future deployment of wideband systems, such as the sub-GHz IEEE standard 802.11ah (which supports channel bandwidths from 1 MHz to 16 MHz) could have significant impact on the license-exempt spectrum. Wideband modulation systems are also subject to the same interference conditions. However, a wideband modulation signal that is co-located with a second wideband signal of different spreading factor or sequence will appear as noise to the target receiver and be treated as such.

Since the on-air duration of a wideband signal can be much longer than that of frequencyhopping narrowband signals we can expect that multiple narrowband signals may well be incident with the wideband modulation, as illustrated below in Figure 3.6.

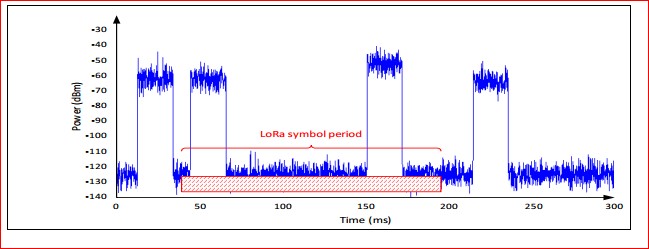


**Figure 3.6: Wideband Signal vs. Narrowband Interferer**

We note that four narrowband signals are incident upon the wideband signal. However, the duration of the narrowband signals are such that in the time domain the interference period is short with respect to the wideband signal.

Due to the redundancy associate with wideband spread-spectrum modulation (recall that each bit

of data is spread of many chips), the modulation is quite resilient to the interference mechanism that appears as bursty short duration pulses. A typical application scenario is illustrated below in Figure 3.7.

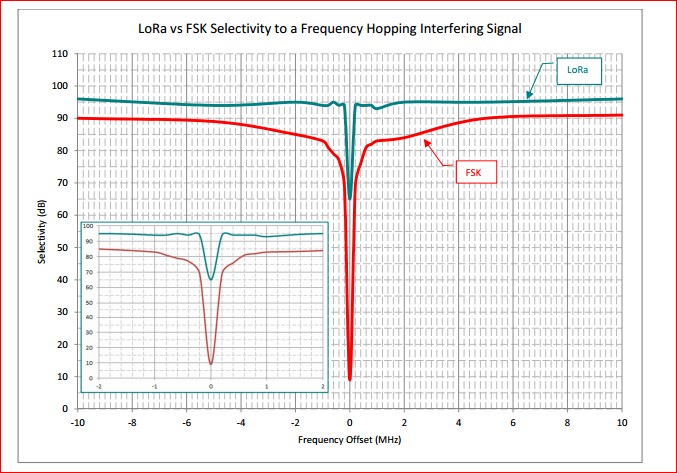


#### Figure 3.7: Example of Burst Interference

Semtech’s LoRa modulation, for example, can tolerate burst interference mechanisms of arbitrary power levels for up to 30% of the symbol length with less than 6 dB sensitivity degradation.

As an example of the robustness of LoRa in the presence of frequency hopping spread-spectrum interference, we can consider the case of an FSK receiver and LoRa receiver of comparable channel bandwidths co-located with an IEEE Standard 802.15.4g transmission which is hopping on a 200 kHz channel raster, as illustrated below in Figure 2.8.

We observe in the case of an adjacent or alternate channel interferer (at ±200 kHz and ±400 kHz, respectively – see inset) that LoRa offers between 15 and 20 dB increased immunity to the unwanted interferer and approximately 10 dB for frequency offsets in excess of 5 MHz (typically, beyond 1 MHz offset, receiver linearity at the expense of current consumption and independent of modulation, dominates).



**Figure 3.8: LoRa vs FSK Selectivity in the vicinity of an AM interfering Signal**

## 3.7 Network Planning Example

### 3.7.1 Capacity

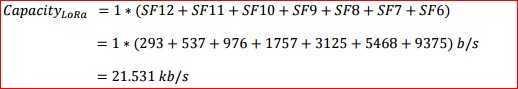
One of the misconceptions concerning the use of spread spectrum wideband modulation is that it is somehow spectral inefficient compared to narrowband modulation. However, consider the case of a narrowband system operating in a virtual channel of 125 kHz bandwidth.

If we assume the case of 12 narrowband FSK channels transmitting at an equivalent bit rate of 1.2 kb/s, then we can calculate the total theoretical channel capacity as:



#### Equation 3.10

If we now consider the same available spectrum deployed as a single 125 kHz LoRa channel, and taking advantage of the orthogonal spreading factors, the equivalent capacity of the channel is now:

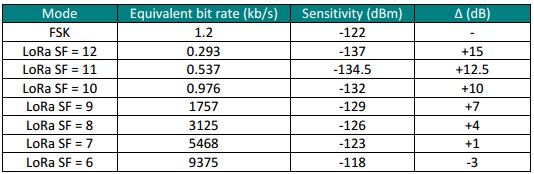


#### Equation 3.11

Thus it can be seen that deploying LoRa modulation provides for a total channel capacity of 21.5 kb/s. This is an increase in channel capacity of nearly 50% compared to FSK.

### 3.7.2 Link Budget

If we now consider the link budget that can be obtained for each modulation scheme and using the example above and assume a sensitivity of -122 dBm that is specified for a conventional FSK transceiver, and compare against that obtainable using LoRa; for a fixed transmitter output power we observe the following link budget delta as tabulated in Table 2.1, below.



#### Table 3.1 link budget delta

Thus we observe that even when transmitting at greater than 4 times the equivalent data rate, LoRa modulation offers similar sensitivity to a conventional FSK system. When the data rate is approximately equivalent the improvement with LoRa is between 7 and 10 dB.

If we consider our channel capacity scenario above, we can see with an equivalent link budget, LoRa can actually transmit a data packet in a quarter of the time required for the FSK system.

Thus assuming a simple TDD or time division multiplexing of the radio channel, LoRa can communication with x4 the number of devices as the FSK system.

### 3.7.3 Throughput Optimization

For a wireless network it can be expected that propagation loss increases with distance from the network coordinator. For narrowband systems this may require additional nodes to be located in a mesh network topology (with increased network complexity and redundancy) or the addition of repeaters for a star network topology to ensure that every device in the network is covered. Unfortunately, the costs associated with installing a repeater can run to between x100 and x1000 the cost of the hardware.

LoRa can minimize this cost by taking advantage of the property that signals with a different spreading factor or sequence will appear as noise at the target receiver. Nodes that are closest to the network coordinator, where path loss allows for transmission at a higher data rate can transmit at the maximum data rate available; as path loss increases with distance the data throughput can be throttled back by increasing the spreading factor or reducing spreading bandwidth.

#### 3.7.3.1 Multi-PHY Mode Networks

For still higher data rates it can be noted that Semtech’s SX127x family of low-power transceiver devices provide for multiple PHY mode operation. Where channel conditions allow, higher data rate FSK modulation can also be employed. As propagation loss increases, LoRa modulation of differing bandwidth and spreading factors can be employed, each transmission tailored to the channel conditions to ensure sufficient link margin. Unlike FSK, LoRa transmissions of differing modulation bandwidth and spreading factors can co-exist.

For dynamic network conditions it is noted that Semtech’s radios can be easily and remotely configured over-the-air.

## 3.8 Conclusions

Semtech’s LoRa modulation is a simple PHY layer implementation that provides significant link budget improvement over conventional narrowband modulation. In addition the enhanced robustness and selectivity provided by the spread spectrum modulation enables greater transmission distance to be obtained, even in harsh, challenging environments.

LoRa modulation uses orthogonal spreading factors. This enables multiple packets of differing spreading factors to be in the same channel concurrently, significantly improving network efficiency and throughput.

Semtech’s family of multi-PHY mode transceivers allow for LoRa to coexist and interoperate with existing legacy network deployments.

# Chapter 4 sx1278 Module

## 4.1 General Description

The SX1276/77/78/79 transceivers feature the LoRaTM long range modem that provides ultralong range spread spectrum communication and high interference immunity whilst minimizing current consumption. Using Semtech’s patented LoRaTM modulation technique SX1276/77/78/79 can achieve a sensitivity of over -148dBm using a low cost crystal and bill of materials. The high sensitivity combined with the integrated +20 dBm power amplifier yields industry leading link budget making it optimal for any application requiring range or robustness. LoRaTM also provides significant advantages in both blocking and selectivity over conventional modulation techniques, solving the traditional design compromise between range, interference immunity and energy consumption. These devices also support high performance (G)FSK modes for systems including WMBus, IEEE802.15.4g. The SX1276/77/78/79 deliver exceptional phase noise, selectivity, receiver linearity and IIP3 for significantly lower current consumption than competing devices.



**Figure 4.1 sx1278 LoRa Module**

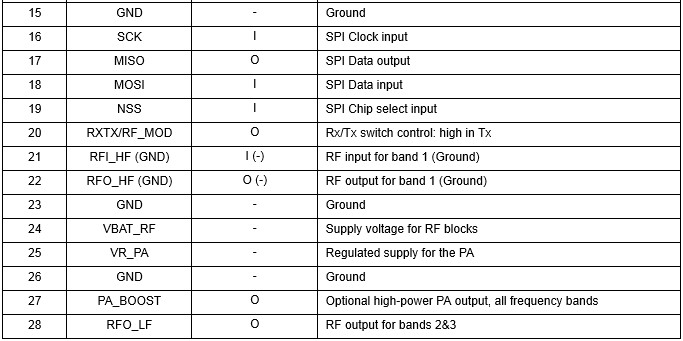
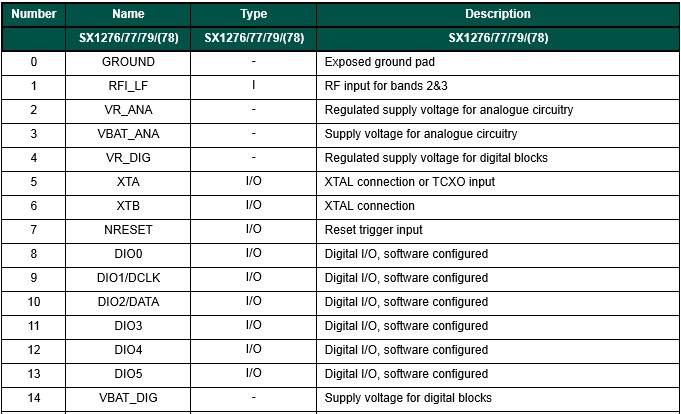
## 4.2 Key Product Features  LoRaTM Modem

* 168 dB maximum link budget
* +20 dBm - 100 mW constant RF output vs. V supply
* +14 dBm high efficiency PA
* Programmable bit rate up to 300 kbps
* High sensitivity: down to -148 dBm
* Bullet-proof front end: IIP3 = -11 dBm
* Excellent blocking immunity
* Low RX current of 9.9 mA, 200 nA register retention
* Fully integrated synthesizer with a resolution of 61 Hz
* FSK, GFSK, MSK, GMSK, LoRaTMand OOK modulation
* Built-in bit synchronizer for clock recovery
* Preamble detection
* 127 dB Dynamic Range RSSI
* Automatic RF Sense and CAD with ultra-fast AFC
* Packet engine up to 256 bytes with CRC
* Built-in temperature sensor and low battery indicator

## 4.3 Electrical Specifications

* Supply Voltage = 3.3v
* Temperature = 25 C
* Fxosc = 32 MHz
* Bandwidth (BW) = 125KHz
* Spreading Factor (SF) = 12
* Error Correction Code (EC) = 4/6
* Packet Error Rate PER = 1%
* CRC on payload enabled
* Output power = 12dbm in Transmission
* Payload Length = 65 Bytes
* Preamble Length = 12 symbols (Program registered preamble length = 8)
* With matched impedances

## 4.4 Pin Description



**Table 4.1 Pin Description of sx1278 chip**

## 4.5 Applications

* Automated Meter Reading.
* Home and Building Automation.
* Wireless Alarm and Security Systems.
* Industrial Monitoring and Control
* Long range Irrigation Systems

# Chapter 5 Project Description

## 5.1 One – to – One Communication

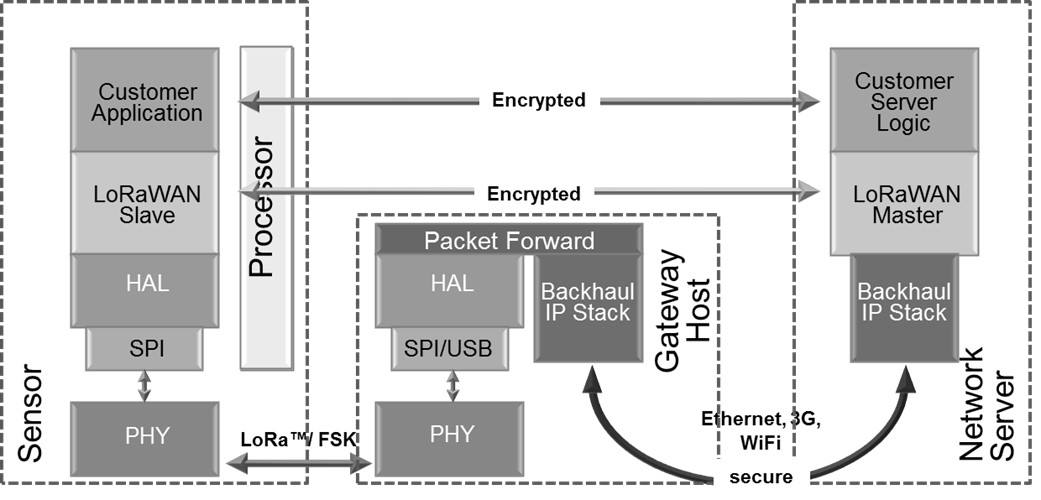
This aim of this project is to simplify and improve farming by providing information to farmers about what they can add to soil in their farms so as to increase production. This can be achieved with the following steps:

* A module containing a microcontroller (Arduino, in this project) connected to LoRa chip (sx1278, in this project) is placed on the farm land. One or more sensors are connected to this circuit. This circuit is called as the ‘end module’.
* These sensors sense various parameters, like moisture or amount of a particular chemical. Apart from these parameters which the circuit senses, there are parameters in this packet which the end module cannot alter, i.e., the packet that gets transmitted contains variables which are both ‘read only’ and ‘write only’ for the end module.
* Such a packet is sent to the Arduino. The board further passes on the information to LoRa chip. The LoRa module transfers this information in the form of a packet.
* Another circuit containing the Arduino and LoRa module is kept around 20km away from the

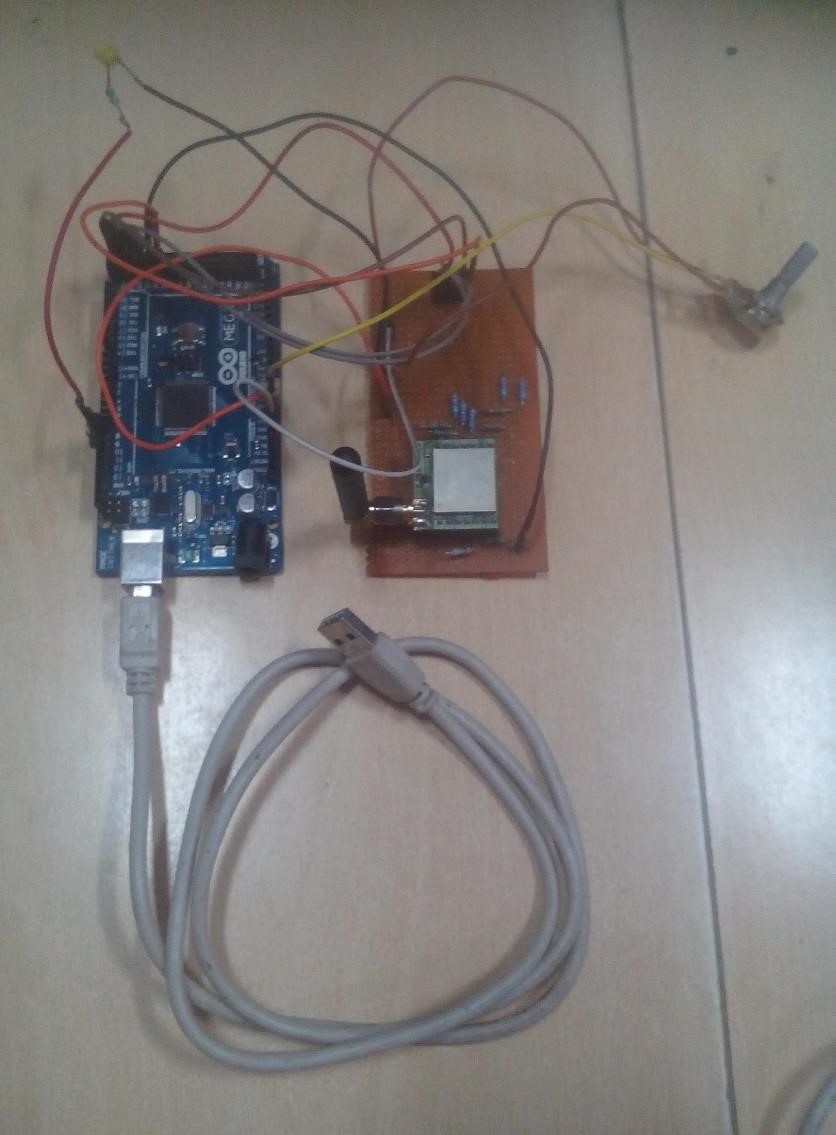
first circuit. This is called as the ‘server’.

* The server captures the packet sent by the end module. It then reads a particular variable(s) (ones that are ‘read only’ for the server). After doing some processing over this(these) variable(s), a data is fed into the ‘write only’ variables.
* The packet is now sent to the Arduino of the server. It is forwarded onto the LoRa chip which is responsible for transmission from server to end module.

While implementing this project we have considered frequency of transmission to be 434MHz, maximum power of +20dBm, spread factor of 12 and LoRa bandwidth of 7.8KHz.



### Figure 5.1 Communication using LoRa



### Figure 5.2 The Setup

However, in this process there is only one end module and a server and by implementing the system like this we are not taking complete benefit of LoRa. This can be achieved by implementing LoRa WAN where several end modules send their packets one at a time to a single server.

## 5.2 LoRa WAN

LoRaWAN is a Low Power Wide Area Network (LPWAN) specification intended for wireless battery operated Things in regional, national or global network. LoRaWAN target key requirements of internet of things such as secure bi-directional communication, mobility and localization services. This standard will provide seamless interoperability among smart Things without the need of complex local installations and gives back the freedom to the user, developer, businesses enabling the roll out of Internet of Things.

LoRaWAN network architecture is typically laid out in a star-of-stars topology in which gateways is a transparent bridge relaying messages between end-devices and a central network server in the backend. Gateways are connected to the network server via standard IP connections while enddevices use single-hop wireless communication to one or many gateways. All end-point communication is generally bi-directional, but also supports operation such as multicast enabling software upgrade over the air or other mass distribution messages to reduce the on air communication time.

Communication between end-devices and gateways is spread out on different frequency channels and data rates. The selection of the data rate is a trade-off between communication range and message duration. Due to the spread spectrum technology, communications with different data rates do not interfere with each other and create a set of "virtual" channels increasing the capacity of the gateway. LoRaWAN data rates range from 0.3 kbps to 50 kbps. To maximize both battery life of the end-devices and overall network capacity, the LoRaWAN network server is managing the data rate and RF output for each end-device individually by means of an adaptive data rate (ADR) scheme.

National wide networks targeting internet of things such as critical infrastructure, confidential personal data or critical functions for the society has a special need for secure communication. This has been solved by several layer of encryption:

* Unique Network key (EUI64) and ensure security on network level
* Unique Application key (EUI64) ensure end to end security on application level  Device specific key (EUI128)

LoRaWAN has several different classes of end-point devices to address the different needs reflected in the wide range of applications:

* Bi-directional end-devices (Class A): End-devices of Class A allow for bi-directional communications whereby each end-device's uplink transmission is followed by two short downlink receive windows. The transmission slot scheduled by the end-device is based on its own communication needs with a small variation based on a random time basis (ALOHA-type of protocol). This Class A operation is the lowest power end-device system for applications that only require downlink communication from the server shortly after the end-device has sent an uplink transmission. Downlink communications from the server at any other time will have to wait until the next scheduled uplink.
* Bi-directional end-devices with scheduled receive slots (Class B): In addition to the Class A random receive windows, Class B devices open extra receive windows at scheduled times. In order for the End-device to open its receive window at the scheduled time it receives a time synchronized Beacon from the gateway. This allows the server to know when the end-device is listening.
* Bi-directional end-devices with maximal receive slots (Class C): End-devices of Class C have nearly continuously open receive windows, only closed when transmitting.

# Chapter 6 Conclusion

This project consists of two separate circuits with the same connections between Arduino board and sx1278chip. One acts as end module and the other as server or controller depending upon the code dumped into the Arduino.

The process begins with the end module sensing serial data input from the potentiometer. The value is stored in a variable called ‘sensorValue’. A check is made to determine whether sensorValue is equal to the data that was previously sensed, i.e., ‘payload1.v1’. Here, payload1 is a structure with elements

‘v1’ and ‘v2’.

If the two are same, then the LoRa module goes into transmit mode and waits for a packet to be delivered to it. If it receives a packet, the two values of payload1 are stored in their respective places and printed on the monitor for the sake of the user. If it receives no packet, then the values that were previously stored in v1 and v2 will be printed. The module is again put into receiver mode.

However, if sensorValue and v1 are different, then the new sensorValue is stored in v1. LoRa module is put into transmit mode and remains there until the variables v1 and v2 are transmitted. It then goes into receiver mode and begins sensing for new value. So, the process continues.

On the other hand, the controller is initially put into receiver mode. If a new packet is received then the variables are stored in v1 and v2 and printed onto the monitor. If no packet is received then the old values are printed again.

Controller then senses if the potentiometer reading, from the potentiometer connected to it. This value is stored in sensorValue and is compared with v2, or previously sensed data. If there is no difference, then the module enters into receiver mode. If a new packet is received then the variables are stored in v1 and v2 and printed onto the monitor. If no packet is received then the old values are printed again.

If sensorValue and v2 are different, then the new sensorValue is stored in v2. LoRa module is put into transmit mode and remains there until the variables v1 and v2 are transmitted. It then goes into receiver mode and begins sensing for new value. So, the process continues.

# Chapter 7 Applications and Future Scope

‘LoRa Communication Protocol for Farming’ is in itself an application of the technology LoRa. So, it would be inappropriate to give any applications to this. However, applications to LoRa technology can be sited in this context. Hence, apart from the few mentioned in section 4.5 here are few more.

LoRa technology is ideal for battery-operated sensor and low-power applications such as:

* Internet of Things
* Smart Agriculture
* Smart City
* Sensor Networks
* Industrial Automation
* Sensor Meters
* Asset Tracking
* Smart Home
* M2M

Furthermore, on March 16, 2016, South Korea’s largest mobile operator, SK Telecom (SKT), plans to deploy a nationwide low-power wide area (LPWA) network this year as part of its long-term strategy to support Internet of Things (IoT) services.

Also, on February 22, 2016, [Semtech Corporation,](http://cts.businesswire.com/ct/CT?id=smartlink&url=http%3A%2F%2Fsemtech.com%2F&esheet=51284174&newsitemid=20160222005498&lan=en-US&anchor=Semtech+Corporation&index=1&md5=eec237795386823e7e2358a7e65f3b6a) a leading supplier of analog and mixed-signal semiconductors, and [ARM,](http://cts.businesswire.com/ct/CT?id=smartlink&url=https%3A%2F%2Fwww.arm.com%2F&esheet=51284174&newsitemid=20160222005498&lan=en-US&anchor=ARM&index=3&md5=222bd0acc35967ef9df824b5cc8a279b) a leader in microprocessor technology, recently released LoRaWAN development through [ARM mbed](http://cts.businesswire.com/ct/CT?id=smartlink&url=https%3A%2F%2Fwww.mbed.com%2Fen%2F&esheet=51284174&newsitemid=20160222005498&lan=en-US&anchor=ARM+mbed&index=5&md5=0a8dab3eeac0e0a9831da8f82f523717) with LoRa-based shields and platforms.

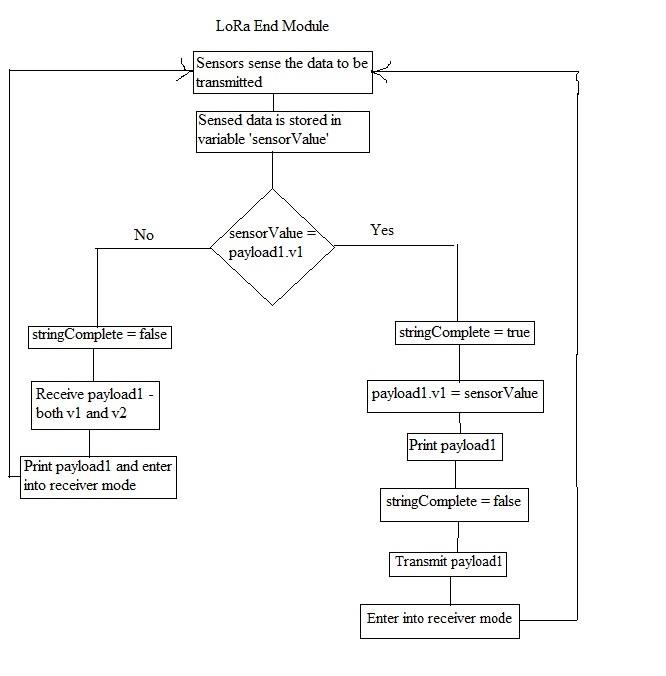
Tata Communications’ proposed LPWAN wireless network is optimized for ultra-low power consumption, which allows the battery in the end device to last for more than a decade without replacement, and has unprecedented reach, enabling communications in deep water and up to 50 meters underground.

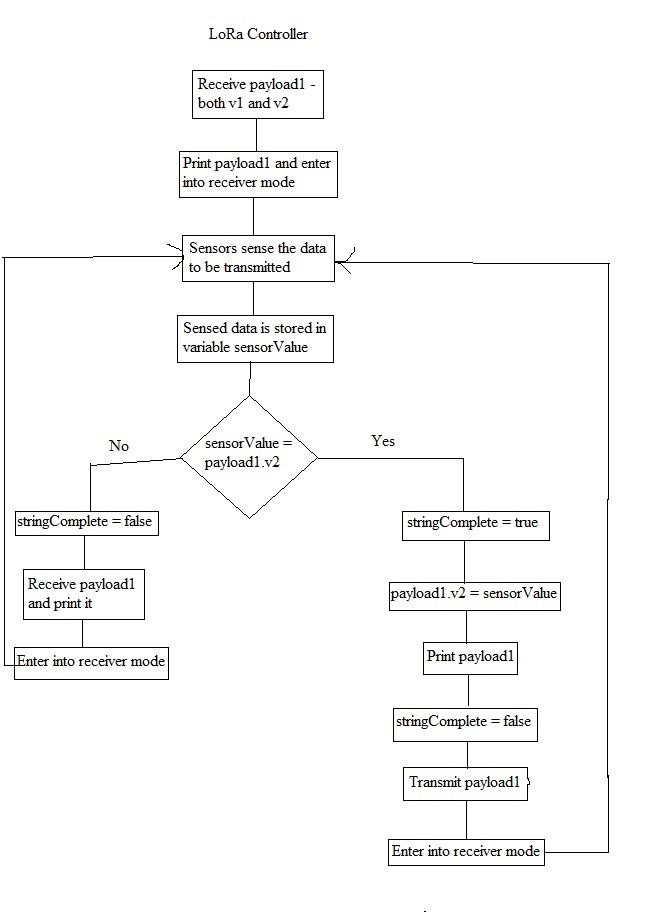
A new wireless sensor network being installed in Japan could help scientists more accurately predict the behavior of the country’s most active volcanoes. The system will gather enormous amounts of data used to forecast volcanic activity, identifying when it might be necessary to issue warnings or evacuations.

The sensor network, which will be installed around 47 volcanoes that the Japanese government has selected for around-the-clock observation, will measure several different variables. In addition to the seismic activity that almost always occurs before an eruption, the sensors will monitor gas emissions, topography changes, and vibrations in the air caused by rocks and ash spewing from the volcano.

These are just a few of those applications which can be completed in near future. Several similar ones are yet to come.

**Flowcharts for Codes:**





**Code:**

/\*End Module\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Project: Pro Micro Arduino + LoRa iMod 4 Server

Data Rate : Determined by program

Frequency: 433Mhz

Modulation: LoRA

Transmit mode : continuous receive

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <SoftwareSerial.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Software Serial Setup

Data Rate : 9600bps

Pinouts: RX (yellow wire) -> TXO (pin 0), TX (red wire) -> RX0, GNd (Blue) -> GND

Transmit mode : continuous transmit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SoftwareSerial console(1, 0); // RX, TX

unsigned char mode; //lora --1 / FSK --0 unsigned char Freq\_Sel; unsigned char Power\_Sel; unsigned char Lora\_Rate\_Sel; unsigned char BandWide\_Sel; unsigned char Fsk\_Rate\_Sel;

// Pinout for Arduino Mega 2560

int led = 13; int nsel = 22; int sck = 24;

int mosi = 26; int miso = 28; int dio0 = 30; int reset = 32; int led1 = 9; // the PWM pin the LED is attached to int brightness = 0; // how bright the LED is

// Define Modes

#define SX1278\_MODE\_RX\_CONTINUOUS 0x00

#define SX1278\_MODE\_TX 0x00

#define SX1278\_MODE\_SLEEP 0x00

#define SX1278\_MODE\_STANDBY 0x00

// System Definitions for SX1278

// Filename : SX1278.h

#define CRC\_4\_5

/\*

#ifdef CR\_4\_5

#define CR 0x01

#else

#ifdef CR\_4\_6

#define CR 0x02

#else

#ifdef CR\_4\_7

#define CR 0x03

#else

#ifdef CR\_4\_8

#define CR 0x04

#endif

#endif

#endif

#endif

\*/

#define CR 0x01

//CRC Enable

#define CRC\_EN

#ifdef CRC\_EN

#define CRC 0x01

#else

#define CRC 0x00

#endif

#define LR\_RegFifo 0x00

#define LR\_RegOpMode 0x01

#define LR\_RegFrMsb 0x06

#define LR\_RegFrMid 0x07

#define LR\_RegFrLsb 0x08

#define LR\_RegPaConfig 0x09

#define LR\_RegPaRamp 0x0A

#define LR\_RegOcp 0x0B

#define LR\_RegLna 0x0C

#define LR\_RegFifoAddrPtr 0x0D

#define LR\_RegFifoTxBaseAddr 0x0E #define LR\_RegFifoRxBaseAddr 0x0F

#define LR\_RegFifoRxCurrentAddr 0x10

#define LR\_RegIrqFlagsMask 0x11

#define LR\_RegIrqFlags 0x12

#define LR\_RegRxNbBytes 0x13

#define LR\_RegRxHeaderCntValueMsb 0x14

#define LR\_RegRxHeaderCntValueLsb 0x15 #define LR\_RegRxPacketCntValueMsb 0x16

#define LR\_RegRxPacketCntValueLsb 0x17 #define LR\_RegModemStat 0x18

#define LR\_RegPktSnrValue 0x19

#define LR\_RegPktRssiValue 0x1A

#define LR\_RegRssiValue 0x1B

#define LR\_RegHopChannel 0x1C

#define LR\_RegModemConfig1 0x1D

#define LR\_RegModemConfig2 0x1E

#define LR\_RegSymbTimeoutLsb 0x1F

#define LR\_RegPreambleMsb 0x20

#define LR\_RegPreambleLsb 0x21

#define LR\_RegPayloadLength 0x22

#define LR\_RegMaxPayloadLength 0x23

#define LR\_RegHopPeriod 0x24

#define LR\_RegFifoRxByteAddr 0x25

#define REG\_LR\_DIOMAPPING1 0x40

#define REG\_LR\_DIOMAPPING2 0x41

#define REG\_LR\_VERSION 0x42

#define REG\_LR\_PLLHOP 0x44 #define REG\_LR\_TCX0 0x4B

#define REG\_LR\_PADAC 0x4D

#define REG\_LR\_FORMERTEMP 0x5B

#define REG\_LR\_AGCREF 0x61

#define REG\_LR\_AGCTHRESH1 0x62

#define REG\_LR\_AGCTHRESH2 0x63

#define REG\_LR\_AGCTHRESH3 0x64

#define RegFIFO 0x00

#define RegOpMode 0x01

#define RegBitRateMsb 0x02

#define RegBitRateLsb 0x03

#define RegFdevMsb 0x04

#define RegFdevLsb 0x05

#define RegFreqMsb 0x06

#define ReqFreqMid 0x07

#define RegFreqLsb 0x08

#define RegPaConfig 0x09

#define RegPaRamp 0x0A

#define RegOcp 0x0B

#define RegLna 0x0C

#define RegRxConfig 0x0D

#define RegRssiConfig 0x0E

#define RegRssiCollision 0x0F

#define RegRssiThresh 0x10

#define RegRssiValue 0x11

#define RegRxBw 0x12

#define RegAfcBw 0x13

#define RegOokPeak 0x14

#define RegOokFix 0x15

#define RegOokAvg 0x16

#define RegAfcFei 0x1A

#define RegAfcMsb 0x1B

#define RegAfcLsb 0x1C

#define RegFeiMsb 0x1D

#define RegFeiLsb 0x1E

#define RegPreambleDetect 0x1F

#define RegRxTimeout1 0x20

#define RegRxTimeout2 0x21

#define RegRxTimeout3 0x22

#define RegRxDelay 0x23 #define RegOsc 0x24

#define RegPreambleMsb 0x25

#define RegPreambleLsb 0x26

#define RegSyncConfig 0x27

#define RegSyncValue1 0x28

#define RegSyncValue2 0x29

#define RegSyncValue3 0x2A #define RegSyncValue4 0x2B

#define RegSyncValue5 0x2C

#define RegSyncValue6 0x2D

#define RegSyncValue7 0x2E

#define RegSyncValue8 0x2F

#define RegPacketConfig1 0x30

#define RegPacketConfig2 0x31

#define RegPayloadLength 0x32

#define RegNodeAdrs 0x33

#define RegBroadcastAdrs 0x34

#define RegFifoThresh 0x35

#define RegSeqConfig1 0x36

#define RegSeqConfig2 0x37

#define RegTimerResol 0x38

#define RegTimer1Coef 0x39

#define RegTimer2Coef 0x3A

#define RegImageCal 0x3B

#define RegTemp 0x3C

#define RegLowBat 0x3D #define RegIrqFlags1 0x3E

#define RegIrqFlags2 0x3F

#define RegDioMapping1 0x40

#define RegDioMapping2 0x41

#define RegVersion 0x42

#define RegPllHop 0x44

#define RegPaDac 0x4D

#define RegBitRateFrac 0x5D

/\*Transmission of WrPara through mosi\*/ void SPICmd8bit(unsigned char WrPara)

{ unsigned char bitcnt;

digitalWrite(nsel, LOW);

digitalWrite(sck, LOW);

for (bitcnt = 8; bitcnt != 0; bitcnt--)

{

digitalWrite(sck, LOW);

if (WrPara & 0x80)

{

digitalWrite(mosi, HIGH);

} else

{

digitalWrite(mosi, LOW);

}

digitalWrite(sck, HIGH);

WrPara <<= 1;

}

digitalWrite(sck, LOW); digitalWrite(mosi, HIGH);

}

/\*Reception from miso into RdPara\*/ unsigned char SPIRead8bit(void)

{ unsigned char RdPara = 0; unsigned char bitcnt;

digitalWrite(nsel, LOW); digitalWrite(mosi, HIGH);

for (bitcnt = 8; bitcnt != 0; bitcnt --)

{

digitalWrite(sck, LOW); RdPara <<= 1; digitalWrite(sck, HIGH);

if (digitalRead(miso))

{

RdPara |= 0x01;

} else

{

RdPara |= 0x00;

}

}

digitalWrite(sck, LOW); return (RdPara);

}

/\*Read data present in adr\*/ unsigned char SPIRead(unsigned char adr)

{

unsigned char tmp; SPICmd8bit(adr); tmp = SPIRead8bit(); digitalWrite(nsel, HIGH); return (tmp);

}

/\*Address and data to be stored in that location are transmitted\*/ void SPIWrite(unsigned char adr, unsigned char WrPara)

{ digitalWrite(nsel, LOW);

SPICmd8bit(adr | 0x80);

SPICmd8bit(WrPara);

digitalWrite(sck, LOW); digitalWrite(mosi, HIGH); digitalWrite(nsel, HIGH);

}

/\*Transmit address and read several data bytes\*/ void SPIBurstRead(unsigned char adr, unsigned char \*ptr, unsigned char leng)

{ unsigned char i; if (leng <= 1)

{ return; } else

{

digitalWrite(sck, LOW); digitalWrite(nsel, LOW);

SPICmd8bit(adr);

for (i = 0; i < leng; i ++ )

{

ptr[i] = SPIRead8bit();

Serial.print(ptr[i]);

}

Serial.println(" "); digitalWrite(nsel, HIGH);

}

}

/\*Transmit address and several data bytes\*/ void BurstWrite(unsigned char adr, unsigned char \*ptr, unsigned char leng)

{ unsigned char i;

if (leng <= 1)

{ return; } else

{

digitalWrite(sck, LOW); digitalWrite(nsel, LOW); SPICmd8bit(adr | 0x80);

for (i = 0; i < leng; i ++)

{

SPICmd8bit(ptr[i]);

Serial.print(ptr[i]);

}

Serial.println(" "); digitalWrite(nsel, HIGH);

}

}

/\*\*\*Parameter Table Definition\*\*\*/

unsigned char sx1278FreqTbl[1][3] =

{

{ 0x6C, 0x80, 0x00 } //434Mhz

};

unsigned char sx1278PowerTbl[4] =

{

0xFF,

0xFC,

0xF9,

0xF6,

};

unsigned char sx1278SpreadFactorTbl[7] =

{

6, 7, 8, 9, 10, 11, 12

};

unsigned char sx1278LoRaBwTbl[10] =

{

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 // 7.8Khz, 10.4KHz, 15.6KHz, 20.8KHz, 31.2KHz,

//41.7KHz, 62.5KHz, 125KHz, 250KHz, 500KHz

};

typedef struct {unsigned char v1[26], v2[26];} payload; payload payload1; boolean stringComplete = false;

/\*Converting from integer to unsigned character array\*/ void toCharArr(int sensorValue)

{ int count = 0; int i = 0; int n = sensorValue; unsigned char arr[26]; while(n != 0) //count = number of digits in sensorValue

{ n /= 10;

++ count;

}

n = sensorValue;

while(count != 0) //conversion

{ arr[i] = (unsigned char)(n / (int)pow (10, count - 1)); n = n % (int)pow (10, count - 1); i ++; count --;

}

while(i < 26) //Appending zeroes

{

arr[i] = 0; i++;

}

for(i = 0; i < 26; i ++)

{ if(arr[i] != payload1.v1[i]) break;

} if(i < 26) //There is new data if i < 26. Transmit it.

{

stringComplete = true; Serial.print("stringComplete = "); Serial.println(stringComplete); for(i = 0; i < 26; i ++)

{ payload1.v1[i] = arr[i];

}

}

}

void AnalogReadSerial()

{

Serial.println("Change potentiometer value to be transmitted"); delay(2000);

int sensorValue = analogRead(A0); // read the input on analog pin 0

toCharArr(sensorValue); //convert into unsigned character array

}

void sx1278\_Standby(void)

{

SPIWrite(LR\_RegOpMode, 0x09); // Standby & Low Frequency mode

}

void sx1278\_Sleep(void)

{

SPIWrite(LR\_RegOpMode, 0x08); // Sleep & Low Frequency mode

}

void sx1278\_EntryLoRa(void)

{

SPIWrite(LR\_RegOpMode, 0x88); // Low frequency mode

}

void sx1278\_LoRaClearIrq(void)

{

SPIWrite(LR\_RegIrqFlags, 0xFF); //Clearing all flags

}

/\*Setting LoRa module in receiver mode\*/ unsigned char sx1278\_LoRaEntryRx(void)

{

unsigned char addr;

Serial.println("Console: Enter sx76 Config"); sx1278\_Config(); // setting base parater

SPIWrite(REG\_LR\_PADAC, 0x84 );

SPIWrite(LR\_RegHopPeriod, 0xFF);

SPIWrite(REG\_LR\_DIOMAPPING1, 0x01 );

SPIWrite(LR\_RegIrqFlagsMask, 0x3f);

sx1278\_LoRaClearIrq();

SPIWrite(LR\_RegPayloadLength, 21);

addr = SPIRead(LR\_RegFifoRxBaseAddr);

SPIWrite(LR\_RegFifoAddrPtr, addr);

SPIWrite(LR\_RegOpMode, 0x8d); // Set the Operating Mode to Continuos Rx Mode && Low Frequency Mode

while (1)

{

if ((SPIRead(LR\_RegModemStat) & 0x04) == 0x04)

{ break; } return 0;

}

}

/\*Read received signal strength indication\*/ unsigned char sx1278\_LoRaReadRSSI(void)

{ unsigned int temp = 10; temp = SPIRead(LR\_RegRssiValue); temp = temp + 127 - 137; return (unsigned char) temp;

}

/\*Begin reception of packet\*/ void RxPacket()

{ unsigned char addr; addr = SPIRead(LR\_RegFifoRxBaseAddr); SPIWrite(LR\_RegFifoAddrPtr, addr);

if (digitalRead(dio0))

{

sx1278\_LoRaRxPacket(0x00, payload1.v1);

delay(6000);

sx1278\_LoRaRxPacket(0x00, payload1.v2);

brightness = map((int)payload1.v2, 0, 10000, 0, 255); analogWrite(led1, brightness); delay(2000);

brightness = 0; analogWrite(led1, brightness);

}

}

/\*Receive one element of payload1\*/ unsigned char sx1278\_LoRaRxPacket (unsigned char adr, unsigned char\* packet)

{

unsigned char i; unsigned char addr; unsigned char packet\_size;

Serial.print("dio0 is ");

Serial.print(dio0, HEX);

Serial.println(" in sx1278\_LoRaRxPacket");

Serial.println("Console: DIO\_0 shows packet recieved");

if (sx1278SpreadFactorTbl[Lora\_Rate\_Sel] == 6 ) //Find packet size

{ packet\_size = 21;

} else

{

packet\_size = SPIRead(LR\_RegRxNbBytes);

}

SPIBurstRead(adr, packet, packet\_size);

sx1278\_LoRaClearIrq();

}

/\*Setting LoRa module into transmit mode\*/ unsigned char sx1278\_LoRaEntryTx(void)

{ unsigned char addr, temp;

sx1278\_Config(); // setting base parater

SPIWrite(REG\_LR\_PADAC, 0x87 );

SPIWrite(LR\_RegHopPeriod, 0x00);

SPIWrite(REG\_LR\_DIOMAPPING1, 0x41 );

sx1278\_LoRaClearIrq();

SPIWrite(LR\_RegIrqFlagsMask, 0xF7);

addr = SPIRead(LR\_RegFifoTxBaseAddr);

SPIWrite(LR\_RegFifoAddrPtr, addr);

SPIWrite(LR\_RegPayloadLength, 21);

while (1)

{

temp = SPIRead(LR\_RegPayloadLength);

Serial.print("temp is ");

Serial.println(temp);

if (temp == 21) break;

}

}

/\*Begin transmission\*/ void TxPacket()

{ unsigned char addr;

addr = SPIRead(LR\_RegFifoTxBaseAddr);

SPIWrite(LR\_RegFifoAddrPtr, addr);

sx1278\_LoRaTxPacket(0x00, payload1.v1);

sx1278\_LoRaEntryRx(); sx1278\_LoRaEntryTx();

sx1278\_LoRaTxPacket(0x00, payload1.v2);

}

/\*Transmit one element of payload1\*/ unsigned char sx1278\_LoRaTxPacket(unsigned char adr, unsigned char\* sxData)

{ unsigned char TxFlag = 0; unsigned char addr; SPIWrite(LR\_RegOpMode, 0x8b);

BurstWrite(adr, (unsigned char \*)sxData, 26);

while (1)

{

Serial.println("In LoRaTxPacket");

Serial.print("dio0 is "); Serial.println(digitalRead(dio0)); if(digitalRead(dio0))

{

sx1278\_LoRaClearIrq(); sx1278\_Standby(); break;

}

}

}

unsigned char sx1278\_ReadRSSI(void)

{ unsigned char temp = 0xff;

temp = SPIRead(0x11); temp >>= 1; temp = 127 - temp; return temp;

}

void sx1278\_Config(void)

{ unsigned char i; sx1278\_Sleep(); // modem must be in sleep mode for(i = 250; i != 0; i --)

{ delay(15);

}

sx1278\_EntryLoRa(); //lora mode

BurstWrite(LR\_RegFrMsb, sx1278FreqTbl[Freq\_Sel], 3); //set the frequency parameter

// set the base parameters

SPIWrite(LR\_RegPaConfig, sx1278PowerTbl[Power\_Sel]); // set the output power parameter

SPIWrite(LR\_RegOcp, 0x0B);

SPIWrite(LR\_RegLna, 0x23);

if (sx1278SpreadFactorTbl[Lora\_Rate\_Sel] == 6)

{ unsigned char tmp;

SPIWrite(LR\_RegModemConfig1, ((sx1278LoRaBwTbl[BandWide\_Sel] << 4) + (CR << 1) +

0x01));

// Implicit Enable CRC Enable (0x02) & Error Coding rate 4/5 (0x01), 4/6 (0x02), 4/7 (0x03), 4/8 (0x04)

SPIWrite(LR\_RegModemConfig2, ((sx1278SpreadFactorTbl[Lora\_Rate\_Sel] << 4) + (CRC << 2) + 0x03));

tmp = SPIRead(0x31); tmp &= 0xF8; tmp |= 0x05;

SPIWrite(0x31, tmp);

SPIWrite(0x37, 0x0C);

}

else

{

Serial.println("config - elseif loop ");

SPIWrite(LR\_RegModemConfig1, ((sx1278LoRaBwTbl[BandWide\_Sel] << 4 ) + (CR << 1) +

0x00));

SPIWrite(LR\_RegModemConfig2, ((sx1278SpreadFactorTbl[Lora\_Rate\_Sel] << 4) + (CRC << 2) + 0x03));

}

SPIWrite(LR\_RegSymbTimeoutLsb, 0xFF);

SPIWrite(LR\_RegPreambleMsb, 0x00);

SPIWrite(LR\_RegPreambleLsb, 12);

SPIWrite(REG\_LR\_DIOMAPPING2, 0x01);

sx1278\_Standby();

Serial.print("Config - finished method, forced Opmde to Standby, opmode is now : ");

Serial.println(SPIRead(LR\_RegOpMode), HEX);

}

void setup()

{

pinMode(led, OUTPUT); pinMode(nsel, OUTPUT); pinMode(sck, OUTPUT); pinMode(mosi, OUTPUT); pinMode(miso, INPUT); pinMode(reset, OUTPUT); pinMode(led1, OUTPUT);

Serial.begin(9600);

Serial.println("Software Serial Port Connected");

Serial.print("OPmode is ");

Serial.print(SPIRead(LR\_RegOpMode), HEX);

Serial.print("\n");

/\*To reset the system\*/ digitalWrite(reset, LOW); delay(0.15); //wait for 150usec digitalWrite(reset, HIGH); delay(10); //wait for 10ms before using the chip

}

void loop()

{

// this is the main code to run repeatedly mode = 0x01; //lora mode Freq\_Sel = 0x00; //433Mhz

Power\_Sel = 0x00;

Lora\_Rate\_Sel = 0x06;

BandWide\_Sel = 0x07;

Fsk\_Rate\_Sel = 0x00;

sx1278\_Config(); sx1278\_LoRaEntryRx();

Serial.println("led high"); digitalWrite(led, HIGH); delay(500); Serial.println("led low"); digitalWrite(led, LOW); delay(500);

int loopCnt = 0;

for (int i = 0; i < 26; i ++)

{ payload1.v1[i] = 0;

}

for (int i = 0; i < 26; i ++)

{ payload1.v2[i] = 0;

}

while (1)

{

AnalogReadSerial(); if (stringComplete)

{

Serial.println("payload1 is: ");

for (int i = 0; i < 26; i ++)

{

Serial.print(payload1.v1[i]);

}

Serial.println(" ");

for (int i = 0; i < 26; i ++)

{

Serial.print(payload1.v2[i]);

}

Serial.println(" ");

stringComplete = false;

// Master

char opMode = SPIRead(LR\_RegOpMode);

Serial.println(" ");

Serial.print("OpMode Reg Value ");

Serial.println(opMode, HEX);

switch (opMode)

{ case 0x08 : case 0x18 : case 0x28 : case 0x38 : case 0x48 : case 0x58 : case 0x68 : case 0x78 : case 0x88 : case 0x98 : case 0xA8 : case 0xB8 : case 0xC8 : case 0xD8 : case 0xE8 : case 0xF8 :

Serial.println("opMode is SLEEP"); break; case 0x09 : case 0x19 : case 0x29 : case 0x39 : case 0x49 : case 0x59 : case 0x69 :

case 0x79 : case 0x89 : case 0x99 : case 0xA9 : case 0xB9 : case 0xC9 : case 0xD9 : case 0xE9 : case 0xF9 :

Serial.println("opMode is STANDBY"); break;

}

Serial.println(" ");

if (opMode != 0x09 || opMode != 0x19 || opMode != 0x29 || opMode != 0x39 || opMode != 0x49 || opMode != 0x59 || opMode != 0x69 || opMode != 0x79 || opMode != 0x89 || opMode != 0x99 || opMode != 0xA9 || opMode != 0xB9 || opMode != 0xC9 || opMode != 0xD9 || opMode != 0xE9 || opMode != 0xF9)

{

SPIWrite(LR\_RegOpMode, B00001001);

Serial.println("Forcing OPMode to STANDBY"); delay(1000);

Serial.print("oPMode is now : ");

Serial.println(SPIRead(LR\_RegOpMode), HEX); delay(3000);

}

Serial.println("led high"); digitalWrite(led, HIGH); delay(200);

Serial.print(loopCnt);

Serial.print(": Check Lora Entry Tx \n");

sx1278\_LoRaEntryTx();

Serial.print(loopCnt);

Serial.print(": Check Lora TX packet \n");

TxPacket();

Serial.println("led low"); digitalWrite(led, LOW);

sx1278\_LoRaEntryRx();

delay(200); loopCnt++;

} else {

// Slave

Serial.println("Console: Waiting for RX packet "); char fromSPI = SPIRead(LR\_RegFifoRxCurrentAddr); Serial.print("Console: Reg FIFO Rx Current Address ");

Serial.print(fromSPI, HEX); Serial.println(" "); delay(1000);

// check the Modem Status Indicators

// Read modem status char lrModemStat = SPIRead(LR\_RegModemStat);

// Signal Detected bit 0

Serial.println();

Serial.print("Console: RegModem (LoRa) Stat 0x"); Serial.print(lrModemStat, HEX);

Serial.println();

SPIWrite(LR\_RegOpMode, 0x85); digitalWrite(dio0, HIGH);

delay(5000);

char showOpMode = SPIRead(LR\_RegOpMode);

if (showOpMode && 0x80 == 0 ) Serial.println("Console: OpMode is FSK"); else

Serial.println("Console: OpMode is LoRa");

showOpMode = SPIRead(LR\_RegOpMode);// && 0xF8;

Serial.print("Console: OpMode Regsiters = ");

Serial.print(showOpMode, HEX);

Serial.println(" ");

char RSSIVal = SPIRead(0x1B); Serial.print("Console: RSSI is ");

Serial.print(RSSIVal, HEX);

Serial.print(" dBm");

Serial.println(" ");

RxPacket();

Serial.println("Console: RX Data buffer contains "); for (int i = 0; i < 26; i ++)

{

Serial.print(payload1.v1[i]);

Serial.print(" ");

}

Serial.println(" ");

for (int i = 0; i < 26; i ++)

{

Serial.print(payload1.v2[i]);

Serial.print(" ");

}

Serial.println(" ");

for (int i = 0; i < 52; i ++)

{

Serial.print("."); delay(10);

}

Serial.println(" ");

sx1278\_LoRaEntryRx();

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Project: Pro Micro Arduino + LoRa iMod 4 Server

Data Rate : Determined by program

Frequency: 433Mhz

Modulation: LoRA

Transmit mode : continuous receive

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <SoftwareSerial.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Software Serial Setup

Data Rate : 9600bps

Pinouts: RX (yellow wire) -> TXO (pin 0), TX (red wire) -> RX0, GNd (Blue) -> GND

Transmit mode : continuous transmit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SoftwareSerial console(1, 0); // RX, TX

unsigned char mode; //lora --1 / FSK --0 unsigned char Freq\_Sel; unsigned char Power\_Sel; unsigned char Lora\_Rate\_Sel; unsigned char BandWide\_Sel; unsigned char Fsk\_Rate\_Sel;

// Pinout for Arduino Mega 2560

int led = 13; int nsel = 22; int sck = 24; int mosi = 26; int miso = 28; int dio0 = 30; int reset = 32; int led1 = 9; // the pin that the LED is attached to int brightness = 0; // how bright the LED is

// Define Modes

#define SX1278\_MODE\_RX\_CONTINUOUS 0x00

#define SX1278\_MODE\_TX 0x00

#define SX1278\_MODE\_SLEEP 0x00

#define SX1278\_MODE\_STANDBY 0x00

// System Definitions for SX1278

// Filename : SX1278.h

#define CRC\_4\_5

/\*

#ifdef CR\_4\_5

#define CR 0x01

#else

#ifdef CR\_4\_6

#define CR 0x02

#else

#ifdef CR\_4\_7

#define CR 0x03

#else

#ifdef CR\_4\_8

#define CR 0x04

#endif

#endif

#endif

#endif

\*/

#define CR 0x01

//CRC Enable

#define CRC\_EN

#ifdef CRC\_EN

#define CRC 0x01

#else

#define CRC 0x00

#endif

#define LR\_RegFifo 0x00

#define LR\_RegOpMode 0x01 #define LR\_RegFrMsb 0x06

#define LR\_RegFrMid 0x07

#define LR\_RegFrLsb 0x08

#define LR\_RegPaConfig 0x09

#define LR\_RegPaRamp 0x0A

#define LR\_RegOcp 0x0B

#define LR\_RegLna 0x0C

#define LR\_RegFifoAddrPtr 0x0D

#define LR\_RegFifoTxBaseAddr 0x0E #define LR\_RegFifoRxBaseAddr 0x0F

#define LR\_RegFifoRxCurrentAddr 0x10

#define LR\_RegIrqFlagsMask 0x11

#define LR\_RegIrqFlags 0x12

#define LR\_RegRxNbBytes 0x13

#define LR\_RegRxHeaderCntValueMsb 0x14

#define LR\_RegRxHeaderCntValueLsb 0x15 #define LR\_RegRxPacketCntValueMsb 0x16

#define LR\_RegRxPacketCntValueLsb 0x17

#define LR\_RegModemStat 0x18

#define LR\_RegPktSnrValue 0x19

#define LR\_RegPktRssiValue 0x1A

#define LR\_RegRssiValue 0x1B

#define LR\_RegHopChannel 0x1C

#define LR\_RegModemConfig1 0x1D

#define LR\_RegModemConfig2 0x1E

#define LR\_RegSymbTimeoutLsb 0x1F

#define LR\_RegPreambleMsb 0x20

#define LR\_RegPreambleLsb 0x21

#define LR\_RegPayloadLength 0x22

#define LR\_RegMaxPayloadLength 0x23

#define LR\_RegHopPeriod 0x24

#define LR\_RegFifoRxByteAddr 0x25 #define REG\_LR\_DIOMAPPING1 0x40

#define REG\_LR\_DIOMAPPING2 0x41

#define REG\_LR\_VERSION 0x42

#define REG\_LR\_PLLHOP 0x44 #define REG\_LR\_TCX0 0x4B

#define REG\_LR\_PADAC 0x4D

#define REG\_LR\_FORMERTEMP 0x5B

#define REG\_LR\_AGCREF 0x61

#define REG\_LR\_AGCTHRESH1 0x62

#define REG\_LR\_AGCTHRESH2 0x63

#define REG\_LR\_AGCTHRESH3 0x64

#define RegFIFO 0x00

#define RegOpMode 0x01

#define RegBitRateMsb 0x02

#define RegBitRateLsb 0x03

#define RegFdevMsb 0x04

#define RegFdevLsb 0x05

#define RegFreqMsb 0x06

#define ReqFreqMid 0x07

#define RegFreqLsb 0x08

#define RegPaConfig 0x09

#define RegPaRamp 0x0A

#define RegOcp 0x0B

#define RegLna 0x0C

#define RegRxConfig 0x0D

#define RegRssiConfig 0x0E

#define RegRssiCollision 0x0F

#define RegRssiThresh 0x10

#define RegRssiValue 0x11

#define RegRxBw 0x12

#define RegAfcBw 0x13

#define RegOokPeak 0x14

#define RegOokFix 0x15

#define RegOokAvg 0x16

#define RegAfcFei 0x1A

#define RegAfcMsb 0x1B

#define RegAfcLsb 0x1C

#define RegFeiMsb 0x1D

#define RegFeiLsb 0x1E

#define RegPreambleDetect 0x1F

#define RegRxTimeout1 0x20

#define RegRxTimeout2 0x21

#define RegRxTimeout3 0x22

#define RegRxDelay 0x23

#define RegOsc 0x24

#define RegPreambleMsb 0x25

#define RegPreambleLsb 0x26

#define RegSyncConfig 0x27

#define RegSyncValue1 0x28

#define RegSyncValue2 0x29

#define RegSyncValue3 0x2A #define RegSyncValue4 0x2B

#define RegSyncValue5 0x2C

#define RegSyncValue6 0x2D

#define RegSyncValue7 0x2E #define RegSyncValue8 0x2F #define RegPacketConfig1 0x30

#define RegPacketConfig2 0x31

#define RegPayloadLength 0x32

#define RegNodeAdrs 0x33

#define RegBroadcastAdrs 0x34

#define RegFifoThresh 0x35

#define RegSeqConfig1 0x36

#define RegSeqConfig2 0x37

#define RegTimerResol 0x38

#define RegTimer1Coef 0x39

#define RegTimer2Coef 0x3A

#define RegImageCal 0x3B

#define RegTemp 0x3C

#define RegLowBat 0x3D

#define RegIrqFlags1 0x3E

#define RegIrqFlags2 0x3F

#define RegDioMapping1 0x40

#define RegDioMapping2 0x41

#define RegVersion 0x42

#define RegPllHop 0x44

#define RegPaDac 0x4D

#define RegBitRateFrac 0x5D

/\*Transmit WrPara through mosi pin\*/ void SPICmd8bit(unsigned char WrPara)

{

unsigned char bitcnt;

digitalWrite(nsel, LOW);

digitalWrite(sck, LOW);

for (bitcnt=8; bitcnt != 0; bitcnt--)

{

digitalWrite(sck, LOW);

if (WrPara&0x80)

{

digitalWrite(mosi, HIGH);

} else

{

digitalWrite(mosi, LOW);

}

digitalWrite(sck, HIGH);

WrPara <<= 1;

}

digitalWrite(sck, LOW); digitalWrite(mosi, HIGH);

}

/\*Reception from miso into RdPara\*/ unsigned char SPIRead8bit(void)

{ unsigned char RdPara = 0; unsigned char bitcnt; digitalWrite(nsel, LOW); digitalWrite(mosi, HIGH);

for(bitcnt = 8; bitcnt != 0; bitcnt --)

{

digitalWrite(sck, LOW); RdPara <<= 1; digitalWrite(sck, HIGH);

if(digitalRead(miso))

{

RdPara |= 0x01;

} else

{

RdPara |= 0x00;

}

}

digitalWrite(sck, LOW); return(RdPara);

}

/\*Read data preasent in adr\*/ unsigned char SPIRead(unsigned char adr)

{ unsigned char tmp; SPICmd8bit(adr); tmp = SPIRead8bit(); digitalWrite(nsel, HIGH); return(tmp);

}

/\*Address and data to be stored in that location are transmitted\*/ void SPIWrite(unsigned char adr, unsigned char WrPara)

{ digitalWrite(nsel, LOW);

SPICmd8bit(adr|0x80);

SPICmd8bit(WrPara);

digitalWrite(sck, LOW); digitalWrite(mosi, HIGH); digitalWrite(nsel, HIGH);

}

/\*Transmit address and read several data bytes\*/ void SPIBurstRead(unsigned char adr, unsigned char \*ptr, unsigned char leng)

{ unsigned char i; if (leng <= 1)

{ return; } else

{

digitalWrite(sck, LOW); digitalWrite(nsel, LOW);

SPICmd8bit(adr);

for (i = 0; i < leng; i ++ )

{

ptr[i] = SPIRead8bit();

Serial.print(ptr[i]);

}

Serial.println(" ");

digitalWrite(nsel, HIGH);

}

}

/\*Transmit address and several data bytes\*/ void BurstWrite(unsigned char adr, unsigned char \*ptr, unsigned char leng)

{ unsigned char i;

if (leng <= 1)

{ return; } else

{

digitalWrite(sck, LOW); digitalWrite(nsel, LOW);

SPICmd8bit(adr|0x80);

for (i = 0; i < leng; i ++)

{

SPICmd8bit(ptr[i]);

Serial.print(ptr[i]);

}

Serial.println(" ");

digitalWrite(nsel, HIGH);

}

}

/\*\*\*\*\*\*\*\*Parameter Table Definition\*\*\*\*\*\*\*\*\*/

unsigned char sx1278FreqTbl[1][3] =

{

{ 0x6C, 0x80, 0x00 } //434Mhz

};

unsigned char sx1278PowerTbl[4] =

{

0xFF,

0xFC,

0xF9,

0xF6,

};

unsigned char sx1278SpreadFactorTbl[7] =

{

6,7,8,9,10,11,12

};

unsigned char sx1278LoRaBwTbl[10] =

{

0,1,2,3,4,5,6,7,8,9 // 7.8Khz, 10.4KHz, 15.6KHz, 20.8KHz, 31.2KHz,

// 41.7KHz, 62.5KHz, 125KHz, 250KHz, 500KHz

};

typedef struct {unsigned char v1[26], v2[26];} payload; payload payload1; boolean stringComplete = false;

/\*Convert the value from integer to unsigned character array\*/ void toCharArr(int sensorValue)

{ int count = 0; int i = 0; int n = sensorValue; unsigned char arr[26]; while(n != 0) // count = number of digits in sensorValue

{ n /= 10; ++ count;

}

n = sensorValue;

while(count != 0) //conversion

{ arr[i] = (unsigned char)(n / (int)pow (10, count - 1)); n = n % (int)pow (10, count - 1); i ++; count --;

}

while(i < 26) //Appending zeroes

{ arr[i] = 0; i ++; }

for(i = 0; i < 26; i ++)

{

if(arr[i] != payload1.v2[i]) break; } if(i < 26) //There is new data if i<26. Transmit it.

{

stringComplete = true; Serial.print("stringComplete = "); Serial.println(stringComplete); for(i = 0; i < 26; i ++)

{ payload1.v2[i] = arr[i];

}

}

}

void AnalogReadSerial()

{

Serial.println("Change potentiometer value to be transmitted"); delay(2000); int sensorValue = analogRead(A0); // read the input on analog pin 0

toCharArr(sensorValue); //convert into unsigned character array

}

void sx1278\_Standby(void)

{

SPIWrite(LR\_RegOpMode, 0x09); // Standby & Low Frequency mode

}

void sx1278\_Sleep(void)

{

SPIWrite(LR\_RegOpMode, 0x08); // Sleep & Low Frequency mode

}

void sx1278\_EntryLoRa(void)

{

SPIWrite(LR\_RegOpMode, 0x88); // Low frequency mode

}

void sx1278\_LoRaClearIrq(void)

{

SPIWrite(LR\_RegIrqFlags, 0xFF); //Clearing all the flags }

/\*setting LoRa module in Receiver mode\*/ unsigned char sx1278\_LoRaEntryRx(void)

{

unsigned char addr;

Serial.println("Console: Enter sx76 Config"); sx1278\_Config(); // setting base parater

SPIWrite(REG\_LR\_PADAC, 0x84 );

SPIWrite(LR\_RegHopPeriod, 0xFF);

SPIWrite(REG\_LR\_DIOMAPPING1, 0x01 );

SPIWrite(LR\_RegIrqFlagsMask, 0x3f);

sx1278\_LoRaClearIrq();

SPIWrite(LR\_RegPayloadLength, 21);

addr = SPIRead(LR\_RegFifoRxBaseAddr);

SPIWrite(LR\_RegFifoAddrPtr, addr);

SPIWrite(LR\_RegOpMode, 0x8d); // Set the Operating Mode to Continuos Rx Mode && Low Frequency Mode

while(1)

{

if ((SPIRead(LR\_RegModemStat) & 0x04) == 0x04)

{

break; } return 0;

}

}

/\*Read received signal strength indication\*/ unsigned char sx1278\_LoRaReadRSSI(void)

{ unsigned int temp = 10; temp = SPIRead(LR\_RegRssiValue); temp = temp + 127 - 137; return (unsigned char) temp;

}

/\*Begin reception of packet\*/ void RxPacket() { unsigned char addr; addr = SPIRead(LR\_RegFifoRxBaseAddr); SPIWrite(LR\_RegFifoAddrPtr, addr); if(digitalRead(dio0))

{

sx1278\_LoRaRxPacket(0x00, payload1.v1);

brightness = map((int)payload1.v1, 0, 10000, 0, 255); analogWrite(led1, brightness); delay(2000);

brightness = 0; analogWrite(led1, brightness);

delay(6000);

sx1278\_LoRaRxPacket(0x00, payload1.v2);

}

}

/\*Receive one elemnet of payload1\*/ unsigned char sx1278\_LoRaRxPacket (unsigned char adr,unsigned char\* packet)

{ unsigned char i; unsigned char addr; unsigned char packet\_size;

Serial.print("dio0 is ");

Serial.print(dio0, HEX);

Serial.println(" in sx1278\_LoRaRxPacket");

Serial.println("Console: DIO\_0 shows packet recieved");

if (sx1278SpreadFactorTbl[Lora\_Rate\_Sel] == 6 )

{ packet\_size = 21;

} else

{

packet\_size = SPIRead(LR\_RegRxNbBytes);

}

SPIBurstRead(adr , packet, packet\_size);

sx1278\_LoRaClearIrq();

}

/\*Setting LoRa module into transmit mode\*/

unsigned char sx1278\_LoRaEntryTx(void)

{ unsigned char addr, temp;

sx1278\_Config(); // setting base parater

SPIWrite(REG\_LR\_PADAC, 0x87 );

SPIWrite(LR\_RegHopPeriod, 0x00);

SPIWrite(REG\_LR\_DIOMAPPING1, 0x41 );

sx1278\_LoRaClearIrq(); SPIWrite(LR\_RegIrqFlagsMask, 0xF7);

addr = SPIRead(LR\_RegFifoTxBaseAddr);

SPIWrite(LR\_RegFifoAddrPtr, addr);

SPIWrite(LR\_RegPayloadLength, 21);

while(1)

{

temp = SPIRead(LR\_RegPayloadLength);

Serial.print("temp is ");

Serial.println(temp);

if(temp == 21) break;

}

}

/\*Begin transmission\*/ void TxPacket() { unsigned char addr; addr = SPIRead(LR\_RegFifoTxBaseAddr);

SPIWrite(LR\_RegFifoAddrPtr, addr);

sx1278\_LoRaTxPacket(0x00, payload1.v1);

sx1278\_LoRaEntryRx(); sx1278\_LoRaEntryTx();

sx1278\_LoRaTxPacket(0x00, payload1.v2);

}

/\*Transmit one element of payload1\*/ unsigned char sx1278\_LoRaTxPacket(unsigned char adr, unsigned char\* sxData)

{ unsigned char TxFlag = 0; unsigned char addr;

SPIWrite(LR\_RegOpMode, 0x8b);

BurstWrite(adr, (unsigned char \*)sxData, 26);

while(1)

{

Serial.println("In LoRaTxPacket");

Serial.print("dio0 is "); Serial.println(digitalRead(dio0)); if(digitalRead(dio0))

{

sx1278\_LoRaClearIrq(); sx1278\_Standby(); break;

}

}

}

unsigned char sx1278\_ReadRSSI(void)

{ unsigned char temp = 0xff;

temp = SPIRead(0x11); temp >>= 1; temp = 127 - temp; return temp;

}

void sx1278\_Config(void)

{ unsigned char i; sx1278\_Sleep(); // modem must be in sleep mode for(i = 250; i != 0; i --)

{ delay(15);

}

sx1278\_EntryLoRa(); //lora mode

BurstWrite(LR\_RegFrMsb, sx1278FreqTbl[Freq\_Sel], 3); //set the frequency parameter

// set the base parameters

SPIWrite(LR\_RegPaConfig, sx1278PowerTbl[Power\_Sel]); // set the output power parameter

SPIWrite(LR\_RegOcp, 0x0B);

SPIWrite(LR\_RegLna, 0x23);

if(sx1278SpreadFactorTbl[Lora\_Rate\_Sel] == 6)

{

unsigned char tmp;

SPIWrite(LR\_RegModemConfig1, ((sx1278LoRaBwTbl[BandWide\_Sel] << 4) + (CR << 1) +

0x01));

// Implicit Enable CRC Enable (0x02) & Error Coding rate 4/5 (0x01), 4/6 (0x02), 4/7 (0x03), 4/8 (0x04)

SPIWrite(LR\_RegModemConfig2, ((sx1278SpreadFactorTbl[Lora\_Rate\_Sel] << 4) + (CRC << 2) + 0x03));

tmp = SPIRead(0x31); tmp &= 0xF8; tmp |= 0x05; SPIWrite(0x31,tmp);

SPIWrite(0x37, 0x0C);

} else

{

Serial.println("config - elseif loop ");

SPIWrite(LR\_RegModemConfig1, ((sx1278LoRaBwTbl[BandWide\_Sel] << 4 ) + (CR << 1) +

0x00));

SPIWrite(LR\_RegModemConfig2, ((sx1278SpreadFactorTbl[Lora\_Rate\_Sel] << 4) + (CRC << 2) + 0x03));

}

SPIWrite(LR\_RegSymbTimeoutLsb, 0xFF);

SPIWrite(LR\_RegPreambleMsb, 0x00);

SPIWrite(LR\_RegPreambleLsb, 12);

SPIWrite(REG\_LR\_DIOMAPPING2, 0x01);

sx1278\_Standby();

Serial.print("Config - finished method, forced Opmde to Standby, opmode is now : "); Serial.println(SPIRead(LR\_RegOpMode), HEX);

}

void setup()

{ pinMode(led, OUTPUT); pinMode(nsel, OUTPUT); pinMode(sck, OUTPUT); pinMode(mosi, OUTPUT); pinMode(miso, INPUT); pinMode(reset, OUTPUT); pinMode(led1, OUTPUT);

Serial.begin(9600);

Serial.println("Software Serial Port Connected");

Serial.print("OPmode is ");

Serial.print(SPIRead(LR\_RegOpMode), HEX);

Serial.print("\n");

/\*To reset the system\*/ digitalWrite(reset, LOW); delay(0.15); //wait for 150usec digitalWrite(reset, HIGH); delay(10); //wait for 10ms before using the chip

}

void loop()

{

// this is the main code to run repeatedly mode = 0x01; //lora mode Freq\_Sel = 0x00; //433Mhz

Power\_Sel = 0x00;

Lora\_Rate\_Sel = 0x06;

BandWide\_Sel = 0x07;

Fsk\_Rate\_Sel = 0x00;

sx1278\_Config(); sx1278\_LoRaEntryRx();

Serial.println("led high"); digitalWrite(led, HIGH); delay(500); Serial.println("led low"); digitalWrite(led, LOW); delay(500);

int loopCnt = 0;

for(int i = 0; i < 26; i ++)

{ payload1.v1[i] = 0;

}

for(int i = 0; i < 26; i ++)

{ payload1.v2[i] = 0;

}

delay(10000);

// Slave

Serial.println("Console: Waiting for RX packet "); char fromSPI = SPIRead(LR\_RegFifoRxCurrentAddr); Serial.print("Console: Reg FIFO Rx Current Address ");

Serial.print(fromSPI, HEX); Serial.println(" "); delay(1000);

// check the Modem Status Indicators

//Read modem status char lrModemStat = SPIRead(LR\_RegModemStat);

// Signal Detected bit 0

Serial.println();

Serial.print("Console: RegModem (LoRa) Stat 0x");

Serial.print(lrModemStat, HEX);

Serial.println();

SPIWrite(LR\_RegOpMode, 0x85); digitalWrite(dio0, HIGH);

delay(5000);

char showOpMode = SPIRead(LR\_RegOpMode);

if(showOpMode && 0x80 == 0 ) Serial.println("Console: OpMode is FSK"); else

Serial.println("Console: OpMode is LoRa");

showOpMode = SPIRead(LR\_RegOpMode);// && 0xF8;

Serial.print("Console: OpMode Regsiters = ");

Serial.print(showOpMode, HEX);

Serial.println(" ");

char RSSIVal = SPIRead(0x1B); Serial.print("Console: RSSI is ");

Serial.print(RSSIVal, HEX);

Serial.print(" dBm");

Serial.println(" ");

RxPacket();

Serial.println("Console: RX Data buffer contains "); for(int i = 0; i < 26; i ++)

{

Serial.print(payload1.v1[i]);

Serial.print(" ");

}

Serial.println(" ");

for(int i = 0; i < 26; i ++)

{

Serial.print(payload1.v2[i]);

Serial.print(" ");

}

Serial.println(" ");

for(int i = 0; i < 52; i ++)

{

Serial.print("."); delay(10);

}

Serial.println(" ");

sx1278\_LoRaEntryRx();

while(1)

{

AnalogReadSerial(); if(stringComplete)

{

Serial.println("payload1 is: "); for(int i = 0; i < 26; i ++)

{

Serial.print(payload1.v1[i]);

}

Serial.println(" ");

for(int i = 0; i < 26; i ++)

{

Serial.print(payload1.v2[i]);

}

Serial.println(" ");

stringComplete = false;

// Master

char opMode = SPIRead(LR\_RegOpMode);

Serial.println(" ");

Serial.print("OpMode Reg Value ");

Serial.println(opMode, HEX);

switch(opMode)

{ case 0x08 : case 0x18 : case 0x28 : case 0x38 : case 0x48 : case 0x58 : case 0x68 : case 0x78 : case 0x88 : case 0x98 : case 0xA8 : case 0xB8 : case 0xC8 : case 0xD8 : case 0xE8 : case 0xF8 :

Serial.println("opMode is SLEEP"); break; case 0x09 : case 0x19 : case 0x29 : case 0x39 : case 0x49 : case 0x59 : case 0x69 : case 0x79 : case 0x89 : case 0x99 : case 0xA9 : case 0xB9 : case 0xC9 : case 0xD9 : case 0xE9 : case 0xF9 :

Serial.println("opMode is STANDBY");

break;

}

Serial.println(" ");

if (opMode != 0x09 || opMode != 0x19 || opMode != 0x29 || opMode != 0x39 || opMode != 0x49 || opMode != 0x59 || opMode != 0x69 || opMode != 0x79 || opMode != 0x89 || opMode != 0x99 || opMode != 0xA9 || opMode != 0xB9 || opMode != 0xC9 || opMode != 0xD9 || opMode != 0xE9 || opMode != 0xF9)

{

SPIWrite(LR\_RegOpMode, B00001001);

Serial.println("Forcing OPMode to STANDBY"); delay(1000);

Serial.print("oPMode is now : ");

Serial.println(SPIRead(LR\_RegOpMode), HEX); delay(3000);

}

Serial.println("led high"); digitalWrite(led, HIGH); delay(200);

Serial.print(loopCnt);

Serial.print(": Check Lora Entry Tx \n");

sx1278\_LoRaEntryTx();

Serial.print(loopCnt);

Serial.print(": Check Lora TX packet \n");

TxPacket();

Serial.println("led low"); digitalWrite(led, LOW);

sx1278\_LoRaEntryRx();

delay(200); loopCnt++;

} else {

// Slave

Serial.println("Console: Waiting for RX packet "); char fromSPI = SPIRead(LR\_RegFifoRxCurrentAddr); Serial.print("Console: Reg FIFO Rx Current Address ");

Serial.print(fromSPI, HEX); Serial.println(" "); delay(1000);

// check the Modem Status Indicators

//Read modem status char lrModemStat = SPIRead(LR\_RegModemStat);

// Signal Detected bit 0

Serial.println();

Serial.print("Console: RegModem (LoRa) Stat 0x");

Serial.print(lrModemStat, HEX);

Serial.println();

SPIWrite(LR\_RegOpMode, 0x85); digitalWrite(dio0, HIGH);

delay(5000);

char showOpMode = SPIRead(LR\_RegOpMode);

if(showOpMode && 0x80 == 0 ) Serial.println("Console: OpMode is FSK"); else

Serial.println("Console: OpMode is LoRa");

showOpMode = SPIRead(LR\_RegOpMode);// && 0xF8;

Serial.print("Console: OpMode Regsiters = ");

Serial.print(showOpMode, HEX);

Serial.println(" ");

char RSSIVal = SPIRead(0x1B); Serial.print("Console: RSSI is ");

Serial.print(RSSIVal, HEX);

Serial.print(" dBm");

Serial.println(" ");

RxPacket();

Serial.println("Console: RX Data buffer contains "); for(int i = 0; i < 26; i ++)

{

Serial.print(payload1.v1[i]);

Serial.print(" ");

}

Serial.println(" ");

for(int i = 0; i < 26; i ++)

{

Serial.print(payload1.v2[i]);

Serial.print(" ");

}

Serial.println(" ");

for(int i = 0; i < 52; i ++)

{

Serial.print("."); delay(10);

}

Serial.println(" ");

sx1278\_LoRaEntryRx();

}

}

}

**References:**

[1]. Semtech Application Note AN1200.13, “SX1272/3/6/7/8: LoRa Modem Designer’s Guide” (http://www.semtech.com/images/datasheet/LoraDesignGuide\_STD.pdf)

[2]. Semtech Application Note AN1200.17, “SX1272/3/6/7/8: LoRa Energy Consumption Design”

(http://www.semtech.com/images/datasheet/LoraLowEnergyDesign\_STD.pdf)

[3]. SX1272 Datasheet (http://www.semtech.com/apps/filedown/down.php?file=sx1272.pdf)

[4]. SX1276 Datasheet (http://www.semtech.com/apps/filedown/down.php?file=sx1276.pdf)

[5]. Rappaport, “Wireless Communications Principals and Practices,” (Prentice-Hall)

[6]. Xiong, “Digital Modulation Techniques, 2nd ed.,” (Artech House)

[7]. Siwak, Bahreini, “Radiowave Propagation and Antennas, ” (Artech House)

[8]. FCC Office of Engineering and Technology Laboratory Division “Guidance for Performing Compliance

Measurements on Digital Transmission Systems (DTS) Operating Under §15.247”

(https://apps.fcc.gov/oetcf/kdb/forms/FTSSearchResultPage.cfm?switch=P&id=21124)

[9]. OFCOM, “Use of Short Range Devices alongside mobile broadband services operating in the 800MHz band”

[10]. CEPT ECC Report 207, Adjacent band co-existence of SRDs in the band 863-870 MHz in light of the LTE usage below 862 MHz

[11]. ERA Technology, “Investigation on the receiver characteristics of SRD equipment in the 863-870 MHz band”

NPSTC, “Public Safety Related Spectrum Issues in the 902-928 MHz Band”