CSE 438-838 Lab 3 IoT System Integration

Deadline: By 8:29 am on September 27, 2024

INTRODUCTION TO LoRa

In this lab, we will integrate a machine-to-machine IoT system that includes periodic sampling, watchdog, run-time error logging, local storage, and LoRa.

What is LoRa and LoRaWAN?

LoRa (short for Long Range) is a spread spectrum modulation technique derived from Chirp Spread Spectrum (CSS) technology. ***LoRaWAN*** is a Long-Range, Low-Power Wide-Area Network (LPWAN) specification designed for the Internet of Things.

LoRa is a proprietary spread spectrum modulation scheme that is a derivative of Chirp Spread Spectrum (CSS) modulation. It 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 to optimize network performance in a constant bandwidth.

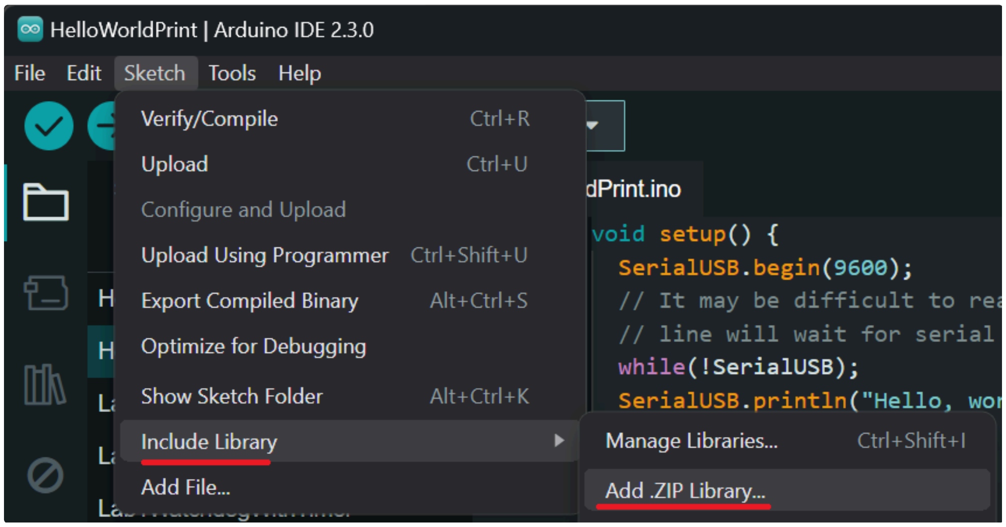
RFM95W LoRa Radio Chip on SparkFun Pro RF

- Point to Point radio capabilities
- LoRa Enabled
- Frequency range: 915 MHz
- Spreading factor: 6-12
- Range up to 1 mile line of sight
- U.FL Antenna

INSTALLING NEW LIBRARIES FOR RFM95W

You will download multiple libraries in this section, and they can all be installed into the Arduino IDE using the following method:

In the Arduino IDE, select `Sketch -> Include Library -> Add .zip Library` and select the downloaded .zip file.


HelloWorldPrint | Arduino IDE 2.3.0
File
Edit
Sketch
Tools
Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Configure and Upload
Upload Using Programmer Ctrl+Shift+U
Export Compiled Binary Alt+Ctrl+S
Optimize for Debugging
Show Sketch Folder Alt+Ctrl+K
Include Library
Add File...
Manage Libraries... Ctrl+Shift+I
Add .ZIP Library...
dPrint.ino

```
void setup() {
  SerialUSB.begin(9600);
  // It may be difficult to rea
  // line will wait for serial
  while(!SerialUSB);
  SerialUSB.println("Hello, wor
```

Installing the RadioHead Library

Download the RadioHead library from its GitHub repository using the link below and then add the library to the Arduino IDE:
https://github.com/PaulStoffregen/RadioHead/archive/master.zip

Installing the FlashStorage Library

Local data storage is needed in many cases. For example, when the radio is duty-cycling to save energy but the data has to be collected, the sensor must store the data first before transmitting it to the gateway. However, local storage, including flash drive/EEPROM, has limited space, and some emergent data that should be forwarded immediately does not need to be stored in the drive.

The FlashStorage library aims to provide a convenient way to store and retrieve user data using the non-volatile flash memory of microcontrollers.
If you would like to read more about it, use this link to go to the GitHub repository page:
https://github.com/cmaglie/FlashStorage

Download the FlashStorage library from its GitHub repository using the link below and then add the library to the Arduino IDE:
https://github.com/cmaglie/FlashStorage/archive/refs/heads/master.zip

Installing the ElectronicCats Library

This library allows you to read the CPU temperature of the SAMD21 using temperature sensors. This gives you an idea of sensing ambient information and collecting data using IoT boards.

You can read more at:
https://github.com/ElectronicCats/ElectronicCats_InternalTemperatureZero

Download the ElectronicCats library from its GitHub repository using the link below and then add the library to the Arduino IDE:
https://github.com/ElectronicCats/ElectronicCats_InternalTemperatureZero/archive/refs/heads/master.zip

CONNECTING THE RADIO

Code Configuration

- Chip Select and Interrupt Pins
  - Pins 12 and 6 are the assigned pins that run to the RM95 Radio Module's chip select and interrupt pins.
  - The pins can be instantiated using `RH_RF95 rf95(12, 6)`
- Frequency select
  - The default frequency in the code is random. It is supposed to be 921.2 but is set a little higher in the code. This falls within the American ISM band of 902-928MHz, but if you're in Europe, that band is 863-870MHz.
  - The frequency can be set using `rf95.setFrequency(frequency)`

Setting Up a Dedicated Channel

To prevent interference, your group should use the equation

```
$Frequency = 902+(CanvasGroupNumber−1)*4$ MHz as the carrier frequency.
```

Machine-To-Machine (M2M) Communication

Machine-To-Machine (M2M) communication is a general concept involving an autonomous device communicating directly to another autonomous device. Autonomous refers to the ability of the node to instantiate and communicate information with another node without human intervention.

The form of communication is left open to the application. An M2M device may use no inherent services or topologies for communication. This leaves out typical internet appliances used regularly for cloud services and storage. An M2M system may also communicate over non-IP-based channels, such as a serial port or custom protocol.

CODE SNIPPETS

Packet Transmission

```
/*
  Both the TX and RX ProRF boards will need a wire antenna. We recommend a 3" piece of wire.
  This example is a modified version of the example provided by the RadioHead
  Library which can be found here:
  www.github.com/PaulStoffregen/RadioHead
*/

#include <SPI.h>

// RadioHead Library:
#include <RH_RF95.h>

// We need to provide the RFM95 module's chip select and interrupt pins to the
// rf95 instance below. On the SparkFun ProRF those pins are 12 and 6 respectively.
RH_RF95 rf95(12, 6);

int LED = 13; // Status LED is on pin 13 (blue)

int packetCounter = 0; // Counts the number of packets sent
long timeSinceLastPacket = 0; // Tracks the time stamp of last packet received

// The broadcast frequency is set to 921.2, but the SAMD21 ProRF operates
// anywhere in the range of 902-928 in the Americas.
// Europe operates in the frequencies 863-870, center frequency at 868MHz.
// This works but it is unknown how well the radio configures to this frequency
float frequency = 915; // Broadcast frequency

void setup()
{
    pinMode(LED, OUTPUT);

    SerialUSB.begin(9600);
    // It may be difficult to read serial messages on startup. The following line
    // will wait for serial to be ready before continuing. Comment out if not needed.
    while (!SerialUSB);
    SerialUSB.println("RFM Client!");

    // Initialize the Radio.
    if (rf95.init() == false) {
        SerialUSB.println("Radio Init Failed - Freezing");
        while (1);
    }
    else {
        // An LED indicator to let us know radio initialization has completed.
        SerialUSB.println("Transmitter up!");
        digitalWrite(LED, HIGH);
        delay(500);
        digitalWrite(LED, LOW);
        delay(500);
    }

    // Set frequency
    rf95.setFrequency(frequency);

    // Transmitter power can range from 14-20dbm.
    rf95.setTxPower(20, false);
}

void loop()
{
    SerialUSB.println("Sending message");
    // Send a message to the other radio
    char toSend[] = "pew";
    packetCounter++;
    SerialUSB.println(toSend);
    SerialUSB.println(packetCounter);

    rf95.send((uint8_t *))toSend, sizeof(toSend));
    // rf95.waitPacketSent();
    delay(1000);
}
```

Note for Packet Transmission:

In void Loop Replace:

```
packetCounter = packetCounter++;
```

with,

```
packetCounter++;
```

To get the packet count to increment, the post-increment operation does not change the value of the variable.

Packet Reception

```
/*
  Both the TX and RX ProRF boards will need a wire antenna. We recommend a 3" piece of wire.
  This example is a modified version of the example provided by the RadioHead
  Library which can be found here:
  www.github.com/PaulStoffregen/RadioHead
*/

#include <SPI.h>

// RadioHead Library:
#include <RH_RF95.h>

// We need to provide the RFM95 module's chip select and interrupt pins to the
// rf95 instance below. On the SparkFun ProRF those pins are 12 and 6 respectively.
RH_RF95 rf95(12, 6);

int LED = 13; // Status LED is on pin 13 (blue)

int packetCounter = 0; // Counts the number of packets sent
long timeSinceLastPacket = 0; // Tracks the time stamp of last packet received

// The broadcast frequency is set to 921.2, but the SAMD21 ProRF operates
// anywhere in the range of 902-928 in the Americas.
// Europe operates in the frequencies 863-870, center frequency at 868MHz.
// This works but it is unknown how well the radio configures to this frequency:
float frequency = 915; // Broadcast frequency

void setup()
{
    pinMode(LED, OUTPUT);

    SerialUSB.begin(9600);
    // It may be difficult to read serial messages on startup. The following line
    // will wait for serial to be ready before continuing. Comment out if not needed.
    while (!SerialUSB);
    SerialUSB.println("RFM Client!");

    // Initialize the Radio
    if (rf95.init() == false) {
        SerialUSB.println("Radio Init Failed - Freezing");
        while (1);
    }
    else {
        // An LED indicator to let us know radio initialization has completed.
        SerialUSB.println("Receiver up!");
        digitalWrite(LED, HIGH);
        delay(500);
        digitalWrite(LED, LOW);
        delay(500);
    }

    // Set frequency.
    rf95.setFrequency(frequency);

    // Transmitter power can range from 14-20dbm.
    rf95.setTxPower(14, true);
}

void loop()
{
    if (rf95.available()) {
        // Should be a message for us now
        uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
        uint8_t len = sizeof(buf);

        if (rf95.recv(buf, &len)) {
            digitalWrite(LED, HIGH); // Turn on status LED
            timeSinceLastPacket = millis(); // Timestamp this packet

            SerialUSB.print("Got message: ");
            SerialUSB.print((char*)buf);
            SerialUSB.print(" RSSI: ");
            SerialUSB.print(rf95.lastRssi(), DEC);
            SerialUSB.println();

            // Send a reply
            // uint8_t toSend[] = "Hello Back!";
            // rf95.send(toSend, sizeof(toSend));
            // rf95.waitPacketSent();
            // SerialUSB.println("Sent a reply");
            // digitalWrite(LED, LOW); //Turn off status LED
        }
        else
            SerialUSB.println("Receive failed");
    }
    // Turn off status LED if we haven't received a packet after 1s
    if (millis() - timeSinceLastPacket > 1000) {
        digitalWrite(LED, LOW); // Turn off status LED
        timeSinceLastPacket = millis(); // Don't write LED but every 1s
    }
}
```

Local Data Storage

The following code snippet is an example of reading and writing Flash.

```
#include <FlashStorage.h>

// Reserve a portion of flash memory to store an "int" variable
// and call it "my_flash_store".
FlashStorage(my_flash_store, int);

// Note: the area of flash memory reserved for the variable is
// lost every time the sketch is uploaded on the board

void() {
    while(!SerialUSB);
    SerialUSB.begin(9600);

    int number;

    // Read the content of "my_flash_store" and assign it to "number"
    number = my_flash_store.read();

    // Print the current number on the serial monitor
    SerialUSB.println(number);

    // Save into "my_flash_store" the number increased by 1 for the
    // next run of the sketch
    my_flash_store.write(number + 1);
}

void loop() {}
```

Temperature Reading

```
/**************************************************************************
  This is a library for internal temperature of the family SAMD
  Electronic Cats invests time and resources providing this open source code,
  please support Electronic Cats and open-source hardware by purchasing products
  from Electronic Cats!   Written by Andrés Sabas Electronic Cats.
  This code is beerware; if you see me (or any other Electronic Cats
  member) at the local, and you've found our code helpful,
  please buy us a round!
  Distributed as-is; no warranty is given.
******************************************************************************/

#include <TemperatureZero.h>

TemperatureZero tempZero = TemperatureZero();

void setup() {
    // put your main code here, to run repeatedly:
    float temperature = TempZero.readInternalTemperature();
    SerialUSB.print('internal Temperature is: ");
    SerualUSB.println(temperature);
    delay(500);
}
```

RUN-TIME ERROR LOGGING

In previous labs, we learned about the importance of run-time error logging. This section will teach you various ways to implement error logging.

CPU Error Logging with WDT

CPU error logging can be done with the ***Device Service Unit (DSU)***. This provides a means to detect debugger probes.

- Enables the ARM Debug Access Port (DAP) to have control over multiplexed debug pads and CPU reset.
- Provides system-level services to debug adapters in an ARM debug system.
- Implements a CoreSight Debug ROM that provides device identification as well as identification of other debug components in the system. Hence, it complies with the ARM Peripheral Identification specification.
- Provides system services to applications that need memory testing.
  - For example, this is required for IEC60730 Class B compliance.
- The DSU can be accessed simultaneously by a debugger and the CPU, as it is connected on the High-Speed Bus Matrix.

See **Section 12** of the **SAMD21 Datasheet** for more details.
Hint: The registers `REG_DSU_STATUSA` and `REG_DSU_STATUSB` will help you debug.

Run-Time Error

There can be multiple types of run-time errors. We consider two examples in this lab;

1. Faulty sensor reading: sensors can get faulty due to external or internal damage. A filter can be used to detect if the sensor data is out of the reasonable range.
2. Communication error: This can be an impaired packet or lost packet due to wireless channel conditions. (Hint: What happens when you disconnect the antennas?)

Information to be Logged

You will need to log the following information for this lab:

1. Timestamps
   a. Timestamp of each entry (if possible)
   b. Count since the last reset
2. System resets
   a. Source of reset (System condition)
3. Run-time errors

   a. Log error codes from run-time functions
   b. Can include failure to allocate memory, stack overflow, communication port data errors, etc.

ASSIGNMENT: COMMUNICATION AND RUN-TIME ERROR LOGGING

You and your teammates will form an M2M network using your nodes and log information and errors.

Requirements

1. **Sensing**
   a. Sense the internal temperature every second.
   b. Implement a sliding window to calculate the average temperature over the last 5 seconds using a stack.
2. **Communication**
   a. Packet structure: Design a packet structure that will include the following information;
      i. Node ID
      ii. Packet ID
      iii. Timestamp
      iv. Payload: sensor data, error log data
   b. Communicate with your teammate's board and transmit your temperature data every 5 seconds so that every board holds a copy of the average temperature of all the nodes.
      - For example, (Alice, temperature value), (Bob, temperature value), (Carter, temperature value).
   c. Elect a node as the leader in the network and send error logs to that board.
   d. Ensure there is a mechanism to avoid collisions. Collisions happen when two or more radio transmitters send their information at the same time. This task will take some planning among team members.
3. **Error-Logging**
   a. *System reset*: Implement a WDT and enable the WDT interrupt, as we learned in Lab 1 and Lab 2.
      - Hints:
        - Enable the WDT timer interrupt handler and read `REG_DSU_STATUSA` and `REG_DSU_STATUSB` on early warning interrupt.
        - Enable and configure WDT early warning interrupt.
   b. *Communication error*:
      i. Packet reception failure (see example code)
      ii. Missing packet
         - Hint: Implement a stack to track packet ID.
   c. *Error log structure*:
      i. Timestamp
      ii. Error code
   d. **Store only the error code in the flash storage**.
4. **Timer**
   For periodical tasks, use the timer technique we learned from lab 2.

Results

1. Code that fulfills each requirement in this lab.
   a. Each function in this system should be separately presented with explanation. Do not submit a snippet of the entire sketch.
2. Serial messages showing:
   a. Timestamps
   b. Packet communication results
   c. Sensor reading results
   d. Average sensor data from other nodes

e. Flash storage results
f. Error logs received on the leader node

Report

1. The requirements for each task
2. Development plan
   - The procedure of solving the problem and the design of your system.
   - Report the configurations used for meeting each requirement in each task in a table, example given below.
   - Report the run-time errors you considered in a table, example table given below.

Register name	Register function	Register value

Error Code	Error

3. Development process
   - Record your development process,
     - **Acknowledge any resources that you found and helped you with your development (open-source projects/forum threads/books)**.
     - Record the software/hardware bugs/pitfalls you had and your troubleshooting procedure
     - Code snippets for each function you develop.
4. Test plan
   - What to test?
     - Each requirement should be tested, e.g., blinking and printing message.
   - What levels of testing?
     - Unit/module level and system level, e.g., for task 2, test two timer modules separately and try them when they are both enabled.
   - Test results of each task should be recorded in a table. Some example tests are shown in the following table but are not limited to these.

Component	Test	Result	Comments
Timer	Normal frequency mode configuration test	fail	Record your observations when you see it fail
Timer	Normal frequency mode configuration test	pass	How did you troubleshoot the previous problem?
Timer	Match frequency mode interrupt test	pass	
LED		not yet run	
System	System test	pass	

5. Results
    - Figures in the report:
        - Screenshots that show you completed the required functions (serial message and Arduino IDE warning).
        - Pictures that show you completed the required functions if necessary.
    - Answer the questions in the assignment.
    - The entire program (As text) in the appendix.

Submission Instructions

1. Submit your lab on Canvas on or before the deadline.
2. Your submission should include one single PDF explaining everything that was asked in the tasks and screenshots, if any.
3. Your submission should also include all the code that you have worked on with proper documentation (Do not attach your code separately as an .ino file. Instead, copy and paste your code in the Appendix. Do not use screenshots in the Appendix.).
4. Failing to follow the instructions will make you lose points.

REFERENCES

1. What is LoRa
   https://www.semtech.com/lora/what-is-lora
2. LoRa modulation basics
   https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf
3. SparkFun SAMD21 Pro RF Hookup Guide:
   https://learn.sparkfun.com/tutorials/sparkfun-samd21-prorf-hookup-guide?_ga=2.127628877.1139230921.1561643965-144910588.1557512622#setting-up-arduino

Deadline: By 8:29 am on September 27, 2024