Fall 2023 - CSCE 438/838: IoT - Lab 3 - IoT System Integration

Introduction

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

1. Introduction to 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 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.

RFM95W LoRa Radio Chip on Sparkfun Pro RF

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

Installing RadioHead library for RFM95W

Radio Head library from its' GitHub repo
https://github.com/PaulStoffregen/RadioHead/archive/master.zip

Adding a .zip library to Arduino

In Arduino IDE select: Sketch > Include Library > Add .zip Library.

Getting Radio connected

- Chip Select and Interupt Pins
  - `**RH_RF95 rf95(12, 6)**` -- Pin 12 and pin 6 are the assigned pins that run to the RM95 Radio Module's chip select and interrupt pins.
- Frequency Select
  - `**rf95.setFrequency(frequency)**` -- The default frequency in the code is kind of random, 921.2 but it falls within the American ISM band: 902-928MHz. If you're in Europe, that band is 863-870MHz. You'll find the variable "frequency" is set a little higher up in the code.

Prevent interference by using dedicated channel

To prevent interference, your group should use the frequency (902+(your group number on Canvas)*4) MHz as the carrier frequency.

Machine-to-machine (M2M) Communication

It 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. It may very well be the case that an M2M device uses no inherent services or topologies for communication. This leaves out typical internet appliances used regularly for cloud services and storage. An M2M system may communicate over non-IP based channels as well, such as a serial port or custom protocol.

Examples

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 Radio Head
  Library which can be found here:
  www.github.com/PaulStoffregen/RadioHeadd
*/

#include <SPI.h>

//Radio Head 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

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 SADM21 ProRf operates
// anywhere in the range of 902-928MHz 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 = 864.1;
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 inidicator to let us know radio initialization has completed.
//    rf95.setModemConfig(Bw125Cr48Sf4096); // slow and reliable?
    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 = packetCounter++;
  SerialUSB.println(toSend);
  SerialUSB.println(packetCounter);

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

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 Radio Head
  Library which can be found here:
  www.github.com/PaulStoffregen/RadioHeadd
*/

#include <SPI.h>

//Radio Head 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

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 SADM21 ProRf operates
// anywhere in the range of 902-928MHz 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 inidicator 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("Recieve 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
  }
}
```

2. Local Data Storage Example

Local data storage is needed for many cases. For example, when radio is duty-cycling to save energy but the data has to be collected, the sensor must store the data first before transmitting 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.

Install the FlashStorage library in your Arduino IDE: https://github.com/cmaglie/FlashStorage The FlashStorage library aims to provide a convenient way to store and retrieve user's data using the non-volatile flash memory of microcontrollers.

You can download the data zip directly from
https://github.com/cmaglie/FlashStorage/archive/refs/heads/master.zip

Example

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 setup() {
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() {}
```

3. Internal Temperature Sensor

Install the Arduino library for internal temperature of the family SAMD. This library helps you to read the CPU temperature of the SAMD21. This gives you an idea of sensing ambient information and collecting data using IoT boards.

https://github.com/ElectronicCats/ElectronicCats_InternalTemperatureZero

You can download the zip directly from:
https://github.com/ElectronicCats/ElectronicCats_InternalTemperatureZero/archive/refs/heads/master.zip

```
/*****************************************************************************
  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 setup code here, to run once:
  SerialUSB.begin(9600);
  TempZero.init();
}

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

4. Run-time Error Logging

We have learnt about run-time error logging and its importance for IoT.

CPU Error Logging with WDT

DSU – Device Service Unit

- The Device Service Unit (DSU) provides a means to detect debugger probes. This enables the ARM Debug Access Port (DAP) to have control over multiplexed debug pads and CPU reset. The DSU also provides system-level services to debug adapters in an ARM debug system. It 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. The DSU also provides system services to applications that need memory testing, as required for IEC60730 Class B compliance, for example. 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 on SAMD21 Datasheet
- REG_DSU_STATUSA and REG_DSU_STATUSB are the registers that 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 a impaired packet or lost packet due to wireless channel condition.

Information to be logged

1. Timestamps
    1. Timestamp of each entry (if possible)
    2. Count since last reset
2. System Resets
    1. Source of reset (System condition)
3. Run-time errors
    1. Log error codes from run-time functions
    2. Failure to allocate mem., stack overflow, communication port data errors, etc.

5. Assignment: Communication and Run-time Error Logging

Requirement

You and your teammates will form a M2M network using your nodes. You will work on the following project together.

1. **Sensing**
    1. Sense the internal temperature every second
    2. Implement a sliding window to calculate the average temperature over the last 5 second using a stack
2. **Communication**
    1. Packet structure: Design a packet structure including the following information
        1. Node ID
        2. Packet ID
        3. Timestamp
        4. Payload: sensor data, error log data
    2. Communicate with your teammate’s board and transmit your temperature data every 5 second such that every board holds a copy of the average temperature of all the nodes, i.e. (Alice, temperature value), (Bob, temperature value), (Carter, temperature value)
    3. Elect a node as the leader in the network and send error log to that board
    4. Ensure that there is a mechanism to avoid collision.
3. **Error-logging**
    1. System reset
        1. Implement a **WDT** and enable the WDT interrupt as we learnt in lab 1 and lab 2
        2. Hint:
            1. Enable WDT timer interrupt handler and read REG_DSU_STATUSA and REG_DSU_STATUSB on early warning interrupt
            2. Enable and configure WDT early warning interrupt
    2. Communication error
        1. Packet reception failure: see example code
        2. Missing packet
            1. Hint: Implement a stack to track packet ID
    3. Error log structure
        1. Timestamp
        2. Error code
    4. *Store only the error code in the flash storage*
4. **Timer**
    1. For periodical tasks, use the timer technique we learnt from last lab.

Results

1. Code that fulfills each requirement in this lab
    1. Each function in this system should be separately presented with explanation, entire code snippet will not be accept
    2. Failing to follow the instruction will make you lose points
2. Serial message showing
    1. Timestamp
    2. Packet communication results
    3. Sensor reading results
    4. Average sensor data from other nodes
    5. Flash storage results
    6. Error logs received on the leader node

Report format

- Report:
  - The requirements for each task
  - Development plan
    - The procedure of solving the problem and the design of your system
    - Report the configurations used for meeting each requirement in a table including:

Register name	Register function	Register value

- Report the run-time errors you considered in a table including:

Error Code	Error

- 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
  - Test Plan
    - What to test?
      - Each requirement should be tested
    - What levels of tests?
      - Unit/module level and system level
    - Test results of each task should be recorded in a table. Some example tests are shown in the following table but tests are not limited to these.

Component	Test	Result	Comment
Timer	Normal frequency mode configuration test	fail	Record your observation when you see it fails
Timer	Normal frequency mode configuration test	pass	How did you troubleshoot and pass the test
Timer	Match frequency mode interrupt test	pass	
LED		not yet run	
System	System test	pass	

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

Submission Instructions:

1. Submit your lab on Canvas on or before the deadline (Sept 15th, 8:29 am)
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
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-pro-rf-hookup-guide?_ga=2.127628877.1139230921.1561643965-144910588.1557512622#setting-up-arduino