**Communication Subsystem for S-Band CubeSat**

by

Zhengxing Liu

Daniel Rea

Rajja Singh

Jiayang Song

(advisor: Jayshri Sabarinathan)

ECE4416 Electrical Engineering Design Project

### Analysis and Test Report

Department of Electrical and Computer Engineering

Western University

London, Ontario, Canada

Date: Friday March 15th, 2019

# 1. Final Analysis of Design Concept

**Excel**

Excel is the main software used for link budget calculation and helped us select our modulation scheme. The modulation scheme part in Excel shows the difference between various digital modulation schemes from which we chose GMSK. GMSK has greater modulation efficiency than BPSK along with better SNR per bit. GMSK also has a simple system and is commonly used in low speed communication which fits perfectly with our requirements.



**MATLAB Antenna Toolbox**

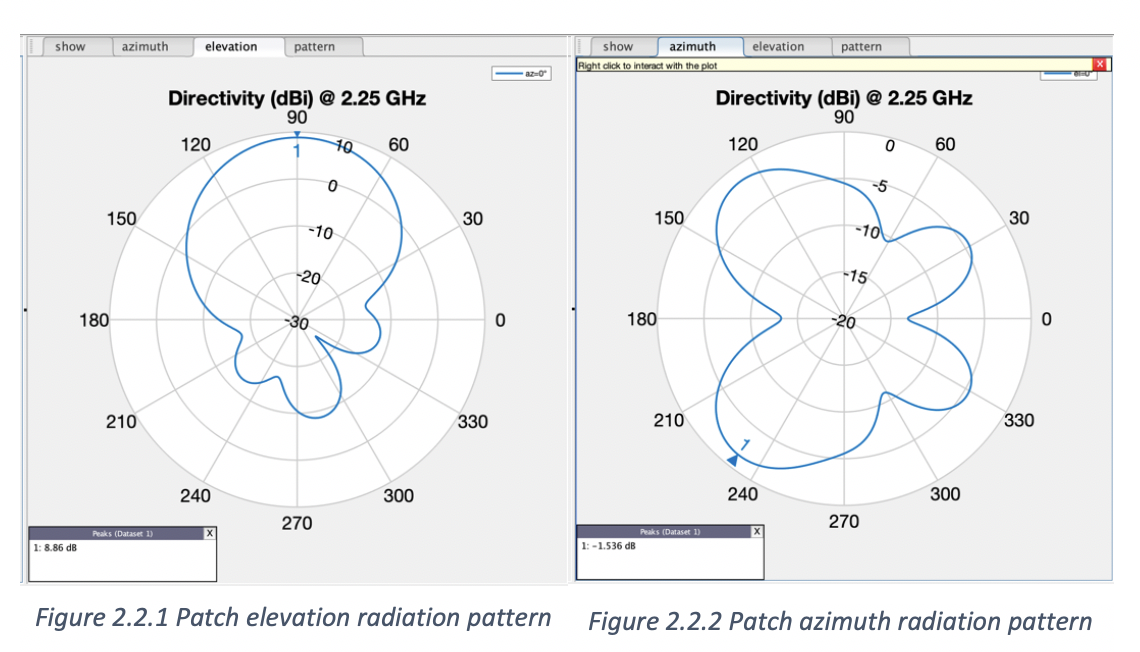
The antenna toolbox is an add-on module in MATLAB which provides simulations and analysis for designing antennas. It computes properties such as impedance, charge distribution and field properties for antenna design. It also provides 3D visualization for the radiation patterns and antenna geometries.

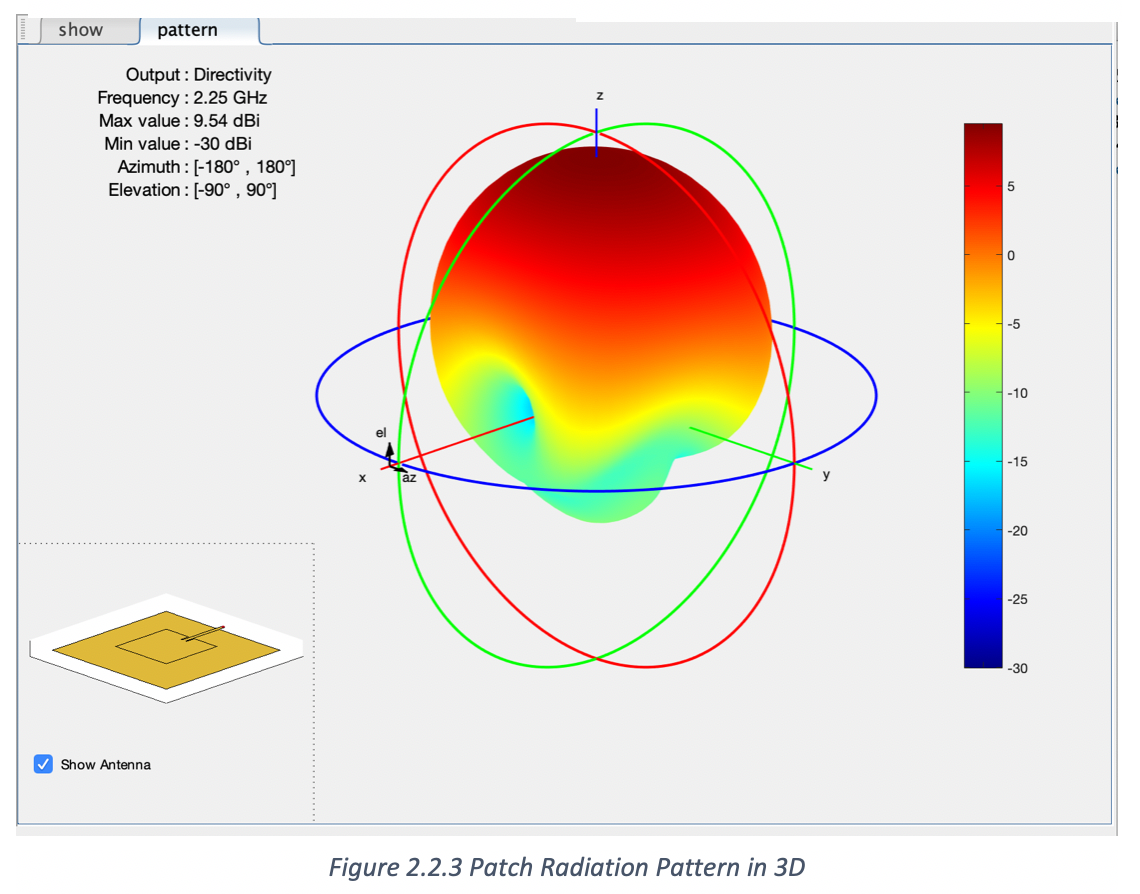
For the purpose of our CubeSat Design, the antenna toolbox is used to determine the radiation pattern for various types of antennas. Models of the antennas are built in the module to generate the 2D (azimuth and elevation) and 3D figures of the radiation patterns. Visual figures help to analyze the antenna gain, beam width and directivity, which are used to evaluation antenna design options.

Since we are designing the transmission link in S-band, the main objective is to achieve a faster data rate at the 2.4GHz frequency rather than the traditional UHF frequency band; therefore, patch antenna is the better option for our design.

Graphically, the patch antenna is more directive than dipole antenna. In the elevation plane, the radiation pattern of the patch antenna creates a distinct separation between the main lobe and side lobes. The antenna steers upwards towards the z-plane with a beam width of 59˚. Because it is more directed to one point, the patch antenna can reach a high maximum gain at 9.53dBi.

**Patch Antenna**





**2. Prototype Fabrication**

The components we had selected for our communication system in the S Band frequencies cost a couple thousand each and it wasn’t feasible to prototype and test with those components in the first year of this project. For our prototype, we went up to the Wifi frequency band and found components to simulate our design that were in our budget.

NRF24L01 Transceiver - $1.70 each

Arduino kit - $50 each

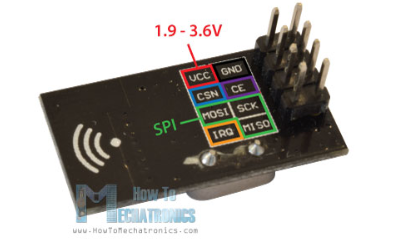


Figure 1: NRF24L01 Transceiver Wiring Diagram

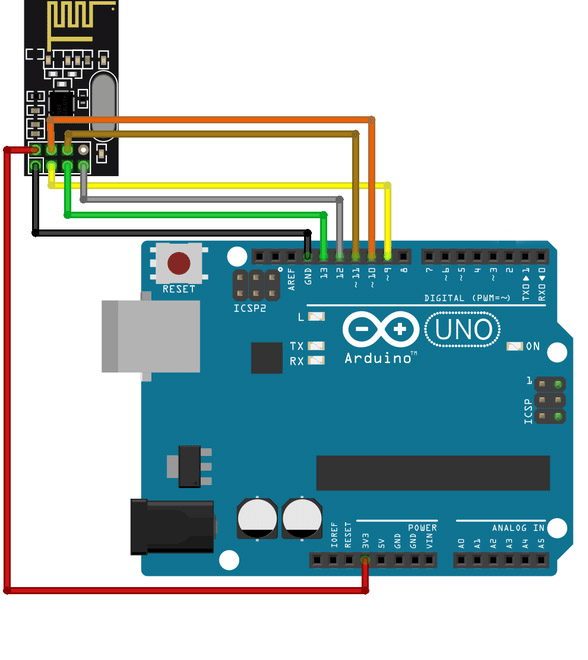


Figure 2: Arduino Uno Wiring Diagram

# A free library from the Arduino website granted students access to all the functions that control the NRF24L01.

# 3. Validation/Testing Strategy or Protocols

Upon obtaining all necessary components to start testing, we took steps to validate our system in a building block manner. The first protocol was to confirm single way communication between the two transceivers. Next came two-way communication validation, which would confirm that each transceiver can both transmit and receive signals through a ‘hand shake’ mechanism. Figure 3 displays the logic implemented.

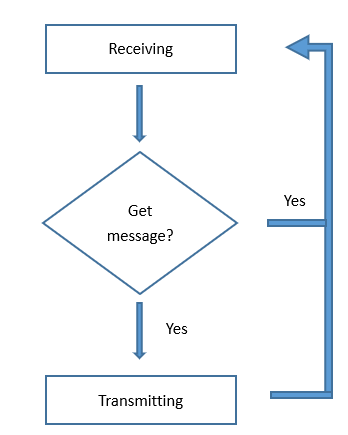
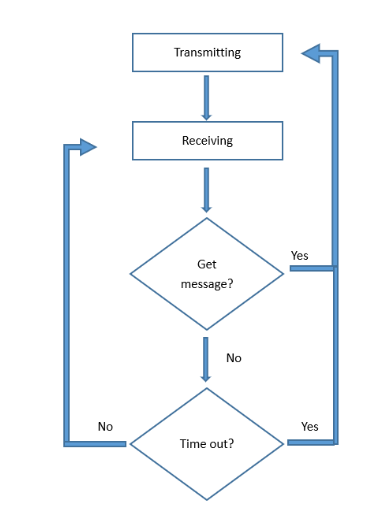


Figure 3: programmed logic for transceiver ‘hand shake’

Performing these tests between the CubeSat and ground station would ensure a communication link is present.

Error rate will determine how efficiently the CubeSat can send and receive data on limited power; it is very important to prioritize. Error rate was determined by sending the same message several times and counting how many times the receiving end sent an acknowledgment back. The difference between those two numbers yielded error rate. Monitoring error rate would allow students to alter the distance and antenna directionality to observe its effects on error rate. All these factors are constantly changing variables that the CubeSat will need to be able to monitor and handle during flight.

**4. Preliminary Validation and Results Analysis**

Single way and two-way communication was relatively simple to achieve. We assigned one transceiver to transmit, and the other to receive, and then swapped their roles. The code used for these tests were from online modules, modified for our needs. Figure 3 are results from the two-way communication.

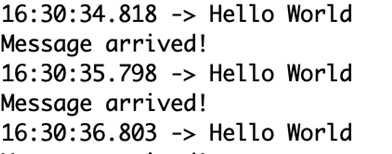


Figure 4:output from transmitter end

This figure shows the message signal “Hello World” being sent from the client, followed by a Tx/Rx role swap for the client to eco confirmation back with “Message arrived!”

Monitoring error rate was a more difficult task to achieve. Sending a string of numbers and having the transceiver compare each character individually proved to be computationally demanding. Instead, one comparison was used to count the number of times a message is sent and received. Figure 5 lists the results.

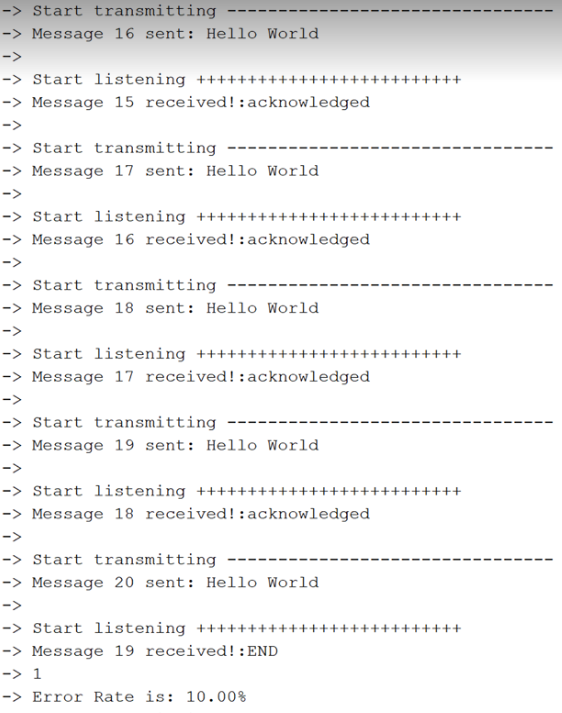


Figure 5: output from client (left) and server (right)

As seen from the client window, after the 20th message was sent, the server replied indicating that this was only the 18th message it had received, thus indicating that 2 message in 20 was not transmitted. The server window offers a concise summary of the success rate. While these test results might differ in magnitude and delivery from the CubeSats complex system, this set up still allows students to test wireless communication, but on a much smaller, cost effective scale.

# 5. Conclusion

Students will continue to add functionality to this prototype to monitor system condition and performance more thoroughly. Response time and received signal power are the next variables to monitor. Distance and antenna directionality will then be altered to observe system response. Achieving these next steps will provide students with hands on experience in understanding wireless communication and dealing with commonly encountered problems.

**6. Appendix-Code Used for Client and Server**

#include <SPI.h> //Begining of Client Code

#include <nRF24L01.h>

#include <RF24.h>

RF24 radio(9, 10); // CE, CSN

const byte addresses[][6] = {"00001", "00002"}; //Byte of array representing the address. This is the address where we will send the data. This should be same on the receiving side.

int time1;

int counter = 0;

int counter2 = 0;

int Error\_rate;

int change;

int response;

int total = 20;

void setup() {

radio.begin(); //Starting the Wireless communication

radio.openWritingPipe(addresses[1]); // 00001

radio.openReadingPipe(1, addresses[0]); // 00002

radio.setPALevel(RF24\_PA\_MIN); //You can set it as minimum or maximum depending on the distance between the transmitter and receiver.

Serial.begin(9600);

Serial.println(F("Start sending------------------------------------------------"));

change = 1;

}

void loop()

{

radio.stopListening(); // This sets the module as transmitter

if (change == 1){ // Change = 1, means we are transmitting

char text1[] = "Hello World";

if(counter2 == total)

{char text3[] = "END";

radio.write(&text3, sizeof(text3));

counter2 = 0;

}

counter2 = counter2 + 1;

radio.write(&text1, sizeof(text1)); //Sending the message to receiver

// time1 = millis();

Serial.println("Start transmitting --------------------------------");

Serial.println("Message " + String(counter2)+ " sent: " + String(text1));

Serial.println();

change = 0; // Change = 0, means we are receiving

}

// Start listening

Serial.println(F("Start listening ++++++++++++++++++++++++++ "));

radio.startListening();

unsigned long time2 = micros();

while(change == 0){

if(radio.available()){

counter = counter+1; // count for # of received message

Serial.print("Message "+ String(counter)+" received!:");

char text2[32] = ""; //Saving the incoming data

radio.read(&text2, sizeof(text2)); //Reading the data

Serial.println(text2);

if (String(text2) == String("END")){

counter = counter - 1;

int res = counter/total;

if(counter < 20)

res = 1;

float Error\_rate = (1 - (float)counter/(total\*(res))\*100;

counter = 0;

Serial.println(res);

Serial.print("Error Rate is: " + String(Error\_rate)+ "%");

}

change = 1;

// response = time2 - time1;

// Serial.print("Response time:");

// Serial.print(response);

Serial.println();

}

if (micros() - time2 > 800000 ){ // If waited longer than 0.8 sec, indicate timeout and exit while loop

Serial.println("Time out!");

change = 1;

break;

}

}

delay(300);

}//End of Client Code

#include <SPI.h> //Beginning of Server Code

#include <nRF24L01.h>

#include <RF24.h>

RF24 radio(9, 10); // CE, CSN

const byte addresses[][6] = {"00001", "00002"};

int count\_msg = 0; //received message counter

int count\_ack = 0; //ack message counter

int total = 20; //number of messages in one set of messages, used for error rate calculation

void setup() {

pinMode(6, OUTPUT);

Serial.begin(9600);

radio.begin();

radio.openWritingPipe(addresses[0]); // 00002

radio.openReadingPipe(1, addresses[1]); // 00001

radio.setPALevel(RF24\_PA\_MIN); //minimum power amplifier

radio.startListening(); //sets the module as receiver

radio.getPALevel();

Serial.println("receiver starts listening");

}

void loop()

{

if (radio.available()) //receiver looking for data

{

char text[32]; //save the incoming message

radio.read(&text, sizeof(text)); //read the message

count\_msg = count\_msg + 1;

Serial.println(String(count\_msg) + " " + String(text));

radio.stopListening(); //send out acknowledge message

char response[] = "acknowledged";

radio.write(&response, sizeof(response));

count\_ack = count\_ack + 1;

if (count\_ack == total) { //notify 20 acknowledges are sent

char endmsg[] = "END";

radio.write(&endmsg, sizeof(endmsg));

count\_ack = 0;

Serial.println("acknowledge END message sent");

}

if (String(text) == String("END")) { //if transmitter has transmitted 20 messages, calculate the error rate

Serial.println("\n-----------------------------------");

count\_msg = count\_msg-1;

int res = count\_msg/total; //if the END message is missed, number of received messages accumulates to next set

if (count\_msg < 20) res = 1;

float succuss = (float)count\_msg/(total\*res)\*100; //error rate calculation

Serial.println(String(count\_msg) + " out of " + String(total\*res) + " messages received\n");

Serial.println(String(float(succuss)) + "% success rate");

Serial.println("-----------------------------------\n");

count\_msg = 0;

}

}

radio.startListening();

delay(500);

}//Ending of Server Code