POLYTECH
LILLE

Rapport du projet de fin d'études

Sujet : Relais Ethernet LoRa

Par :

Cong CHEN
Sonia NDUWAYO

IMA5
Année : 2016-2017

Tuteur : ***Mr. Xavier REDON***
Professeur à PolyTech'Lille

Remerciements

Nous exprimons nos sincères remerciements à notre tuteur de projet Mr.Xavier REDON pour la qualité de son encadrement et le suivi durant tout le déroulement du projet,sa disponibilité.

Nous remercions également les encadrants Mr.Alexandre BOE et Mr. Thomas VANTROYS pour leurs conseils précieux.

Nous adressons également un grand merci à Mr.Thierry FLAMEN pour les suggestions et son aide dans de nombreuses situations surtout en matière de soudure et de design de cartes électroniques.

Nous tenons à remercier aussi Mr. Laurent ENGELS pour sa patience et pour la vidéo du projet.

I. Cahier des charges

1. Contexte

Actuellement, les réseaux personnels dépendent fortement des opérateurs privés.
L'idée de ce projet est de créer un réseau personnel basse consommation et longue portée.

2. Objectif

L'objectif de notre projet est de réaliser un module connectable sur une box Internet permettant de relier deux sites avec une connexion longue distance. Nous avons utilisé le module inAir9(SX1276) basé sur la technologie LoRa de chez *SemTech* pour créer une communication longue portée et la carte réseau ENC28J60 de chez *Microchip* pour préparer, envoyer et contrôler les données sur le réseau.

3. Méthodes

Afin de mener à bien notre projet , nous aurons besoin du matériel déjà existant tel que :

- Le prototype Arduino (micro-contrôleur AtMega 328P),
- le prototype Jeelabs (EtherCard),
- et le prototype inAir9.

Nous pourrions ensuite dans le futur, réaliser un prototype avec ARM Cortex.

II. Travail réalisé

1. Tests effectués

Dans un premier temps, la partie réseau et la partie communication sans fil, gérées respectivement par la carte ENC28J60 et le module LoRa SX1276 ont été développées et testées séparément.

A. Test de l'ENC28J60

Pour la partie carte réseau, le travail consiste à faire en sorte de de recevoir des données, envoyer des données via la liaison série TX/RX et récupérer des données(les données sont stockées dans la mémoire de celle-ci).
Les manipulations ont été faites entre deux PC à l'aide de deux cartes réseau et de deux cartes Arduino.

Ci dessous, schéma représentatif de la connexion:

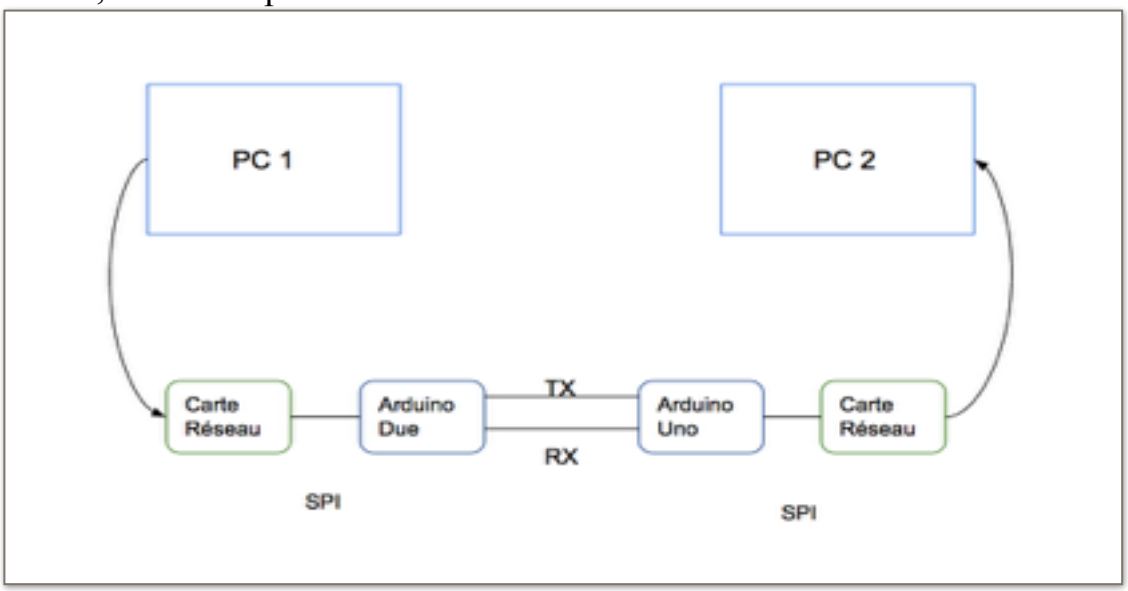


Fig.1 : Réalisation d'un 'réseau local'

Avec quelques modifications de la bibliothèque de la carte réseau, nous avons réussi à lire directement un octet d'un paquet Ethernet dans le buffer de l'ENC28J60 et l'envoyer sur le port série sans enregistrer les données dans la mémoire d'Arduino. Et de l'autre coté, la carte réseau reçoit les octets et les transmet à l'autre PC.

Pour visualiser les paquets qui transitent sur le « réseau local », l'outil de capture tel

tcpdump permet de les analyser en direct.

Sur la capture d'écran, on peut plus ou moins voir quelques informations: type de paquet, TTL, time, longueur, …..

Fig.2 :Ping d'une machine A vers B

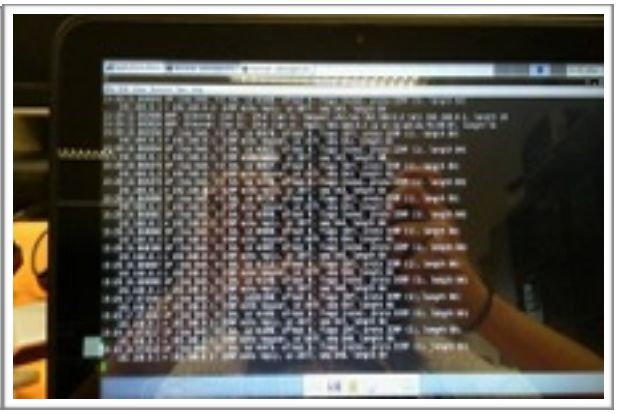


Fig.3 : Capture des paquets Ethernet par tcpdump

B. Test du LoRa

La communication sans fil du prototype est gérée par les modules inAir9.
Ces derniers sont faits avec des composants de type CMS, petits et légers, mais avec l'avantage d'une faible consommation d'énergie et sont munis d'un connecteur SMA pour brancher une antenne.
Ces modules sont connectés à l'Arduino (avec micro-contrôleur AtMega328P) via l'interface série SPI.
Ci dessous un schéma illustrant le prototype :

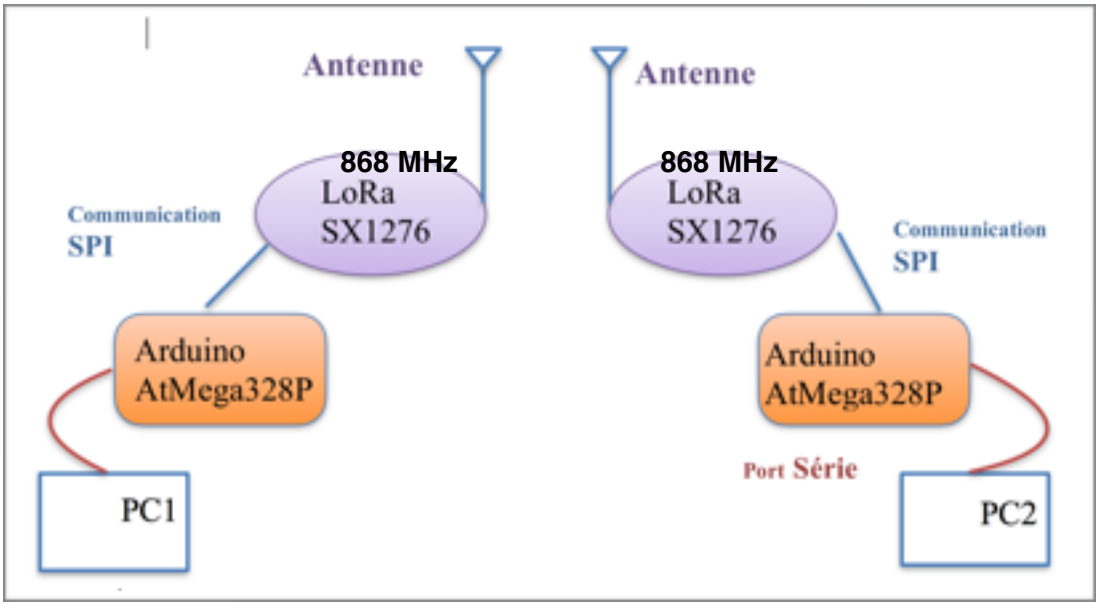


Fig.4 : Réalisation d'une communication sans fil

Le programme implémenté a été écrit en langage bas niveau, le « C ».
L'intérêt majeur d'utiliser ce langage est le fait qu'il soit le plus adapté à la programmation des micro-contrôleurs mais aussi et surtout que le code est très optimisé ainsi que la vitesse d'exécution.
Ceci signifie qu'on peut écrire ou lire directement dans les registres du moduleSX1276 via SPI.

Le programme de test uniquement pour la communication est décomposé en deux parties :

* Une partie « initialisation » : qui sert à configurer les registres du SX1276.
* Une partie « main » : qui s'exécute en permanence et qui permet de contrôler la communication dès qu'il y a réception ou envoi de fragments de paquets.

Les attentes actives n'étant pas une solution idéale car elles immobilisent le processeur uniquement pour attendre un événement, restreignent aussi l'accès du bus mémoire et ralentissent les autres processus,nous avons donc décidé d'utiliser les interruptions pour agir lorsqu'il y a un changement d'état sur la broche d'interruptions si un paquet est prêt d'être envoyé ou reçu.

Plusieurs paramètres influent sur la communication LoRa à savoir : le facteur d'étalement (indicateur sur la quantité de données redondantes qui sont transmises). Plus le facteur d'étalement est élevé, moins sera le débit de données à transmettre, la bande passante, le débit de données,..
En modulation LoRa, la taille maximale d'un paquet est de 255 octets. Cependant, en raison des restrictions de transmission dans l'air, la taille maximale des paquets peut être plus petite, ce qui nous a conduit à fragmenter les paquets LoRa(car la taille maximale de 255 octets est nettement inférieure à la taille maximale d'un paquet Ethernet, pouvant aller jusqu'à 1500 octets).

Après plusieurs tests, nous avons pu établir une communication bas débit entre les 2 plateformes. Ci dessous des images illustrant :

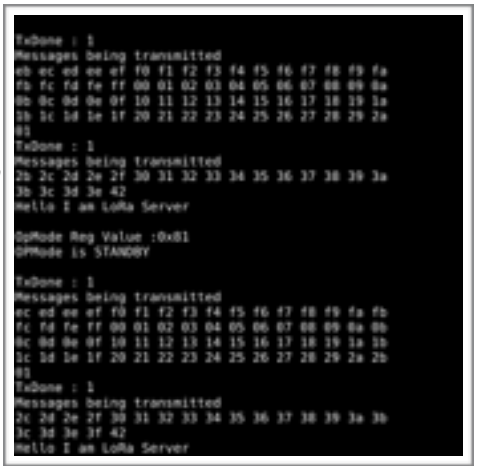


Fig.5 :Transmission LoRa1

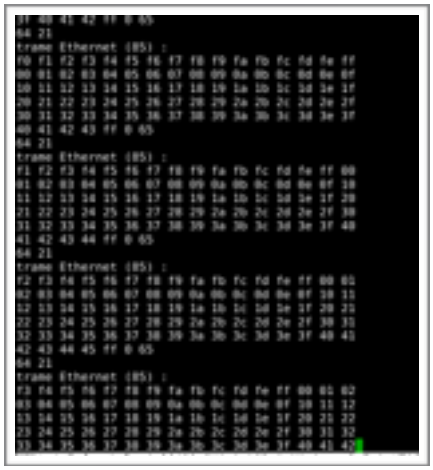


Fig.6 : Réception LoRa1

C. Communication entre les 2 plateformes

Maintenant que nous sommes certains que les communications sont possibles, nous pouvons donc essayer de transmettre les paquets,reçus sur le réseau (Ethernet), par le LoRa. Voici le schéma global :

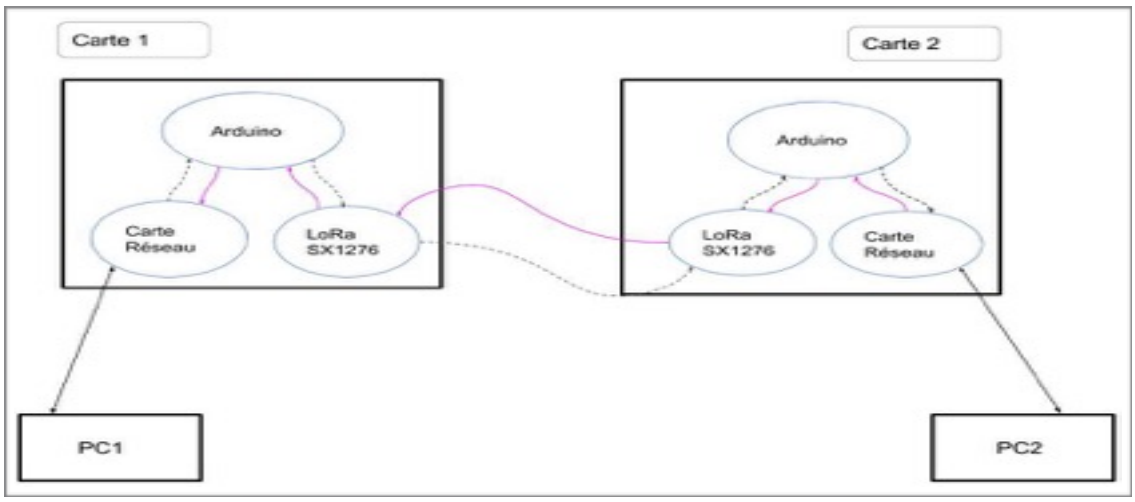
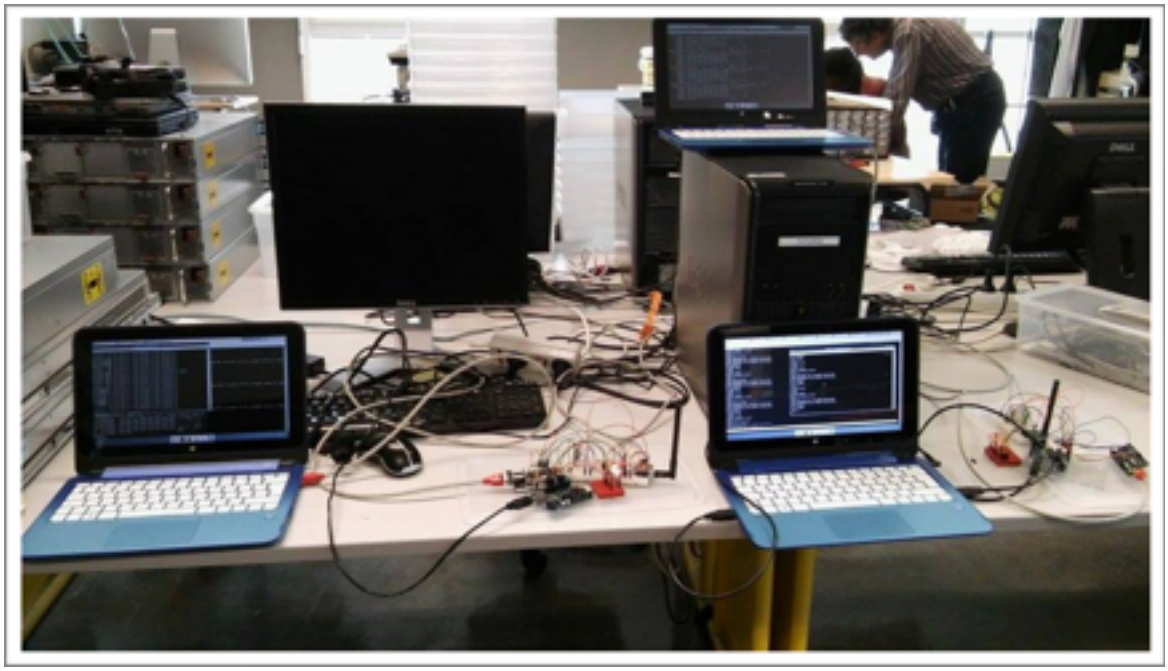


Fig.6 : Relai Ethernet LoRa

La mise en commun des 2 prototype nous a permis de réaliser la plateforme Ethernet - LoRa et ensuite de tester l'algorithme d'émission/réception.



Remarquons ici que les 2 plateformes peuvent communiquer en « full duplex ».(voir vidéo sur le wiki).

Pour visualiser mieux nous avons ajouté des Leds de debuggage.

- La LED rouge clignote lorsqu'un paquet LoRa est transmis,
- la LED orange clignote lorsqu'un paquet LoRa est reçu.

Les tests ont été concluants; les Leds clignotaient alternativement sur les 2 plateformes. Nous avons pu observé avec ***tcpdump*** la réception des paquets ICMP et des paquets ARP via le LoRa.

Nous avons aussi effectué d'autres tests sur de longues distances.

D. Développement du code

Tous les fichiers sources sont écrits en C.

- Nous avons programmé la carte réseau ENC28J60 et le module SX1276 via SPI par Arduino.
- La librairie SPI a été réécrite en accédant directement aux registres. Elle a été fusionnée avec la librairie SPI de l'ENC28J60.
- Afin de simplifier l'écriture du programme principal, nous avons structuré le projet en sous-répertoires(chaque sous-répertoire contient un fichier source .c, le header .h, et une librairie .a) et un Makefile qui gère la compilation,les liens et les dépendances entre les bibliothèques.

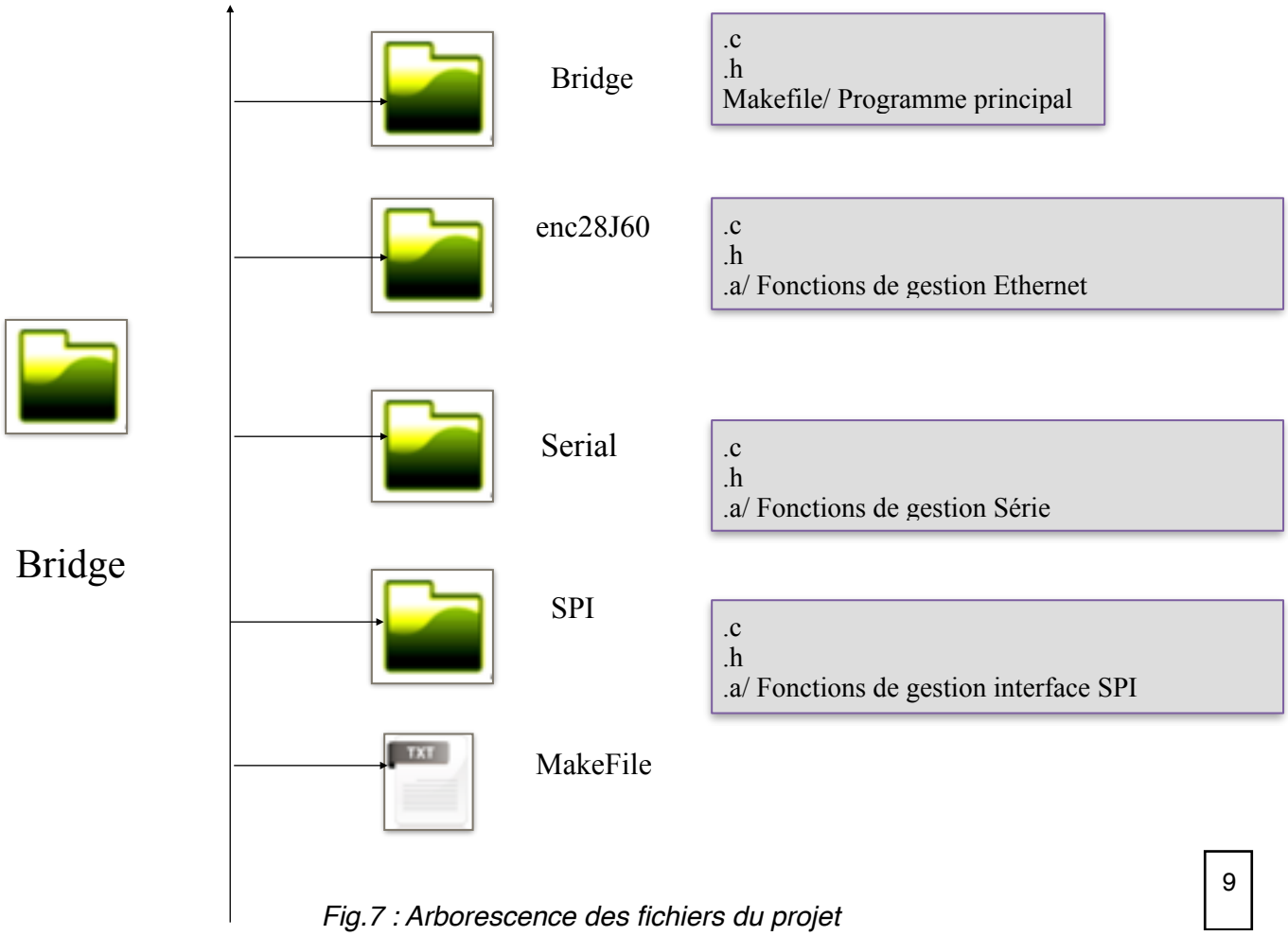


Fig.7 : Arborescence des fichiers du projet

E. Algorithme fonctionnel

Cependant, l'Ethernet étant capable de communiquer en full duplex et Le module SX1276 en mode half duplex(c'est à dire qu'il ne peut pas recevoir et envoyer en même temps), un risque de collision peut se produire sur le réseau.

Voici un algorithme que nous avons essayé d'implémenter pour tenter d'y remédier :

```
Boucle
  Si pas de paquet Ethernet en stock Alors
        tenter de lire un paquet Ethernet pour le mettre en stock
  FSi
  Si pas de réception en cours Alors
   Si un paquet Ethernet en stock Alors
        envoyer un fragment LoRA
    Si dernier fragment Alors
        libérer la mémoire du paquet Ethernet
        attendre un paquet LoRa un temps raisonnable
    Fsi
    Sinon
    Si un paquet Ethernet vient d'être reçu par LoRa Alors
      envoyer un paquet LoRa de taille 1
   FSi
  Fsi
  Si pas d'envoi en cours et paquet LoRa reçu Alors
    Si paquet est un fragment (taille > 1) Alors
    stocker le fragment dans le tampon d'envoi Ethernet
    Si dernier fragment Alors
     envoyer le paquet sur Ethernet
    FSi
   FSi
  FSi
 FinBoucle
```

Fig.8: Algorithme du Relai Ethernet LoRa

Dans cet algorithme, l'émetteur envoie un paquet et attend une réponse du récepteur. Une fois qu'il y a une réponse de retour, l'émetteur peut continuer à émettre.
Au cas où le récepteur reçoit des données mais qu'il n'a rien à renvoyer, il envoie un seul octet pour débloquer l'émetteur. Cela permet de résoudre le problème de half-duplex.

F. Réalisation des cartes

- *Logiciel Altium* :

Avec le logiciel Altium, nous avons designé une carte électronique
Pour cela, nous avons réalisé les empreintes des composants non disponibles ainsi que le design de l'antenne(sur Eagle).
Voici un exemple d'une empreinte du composant PE4259:

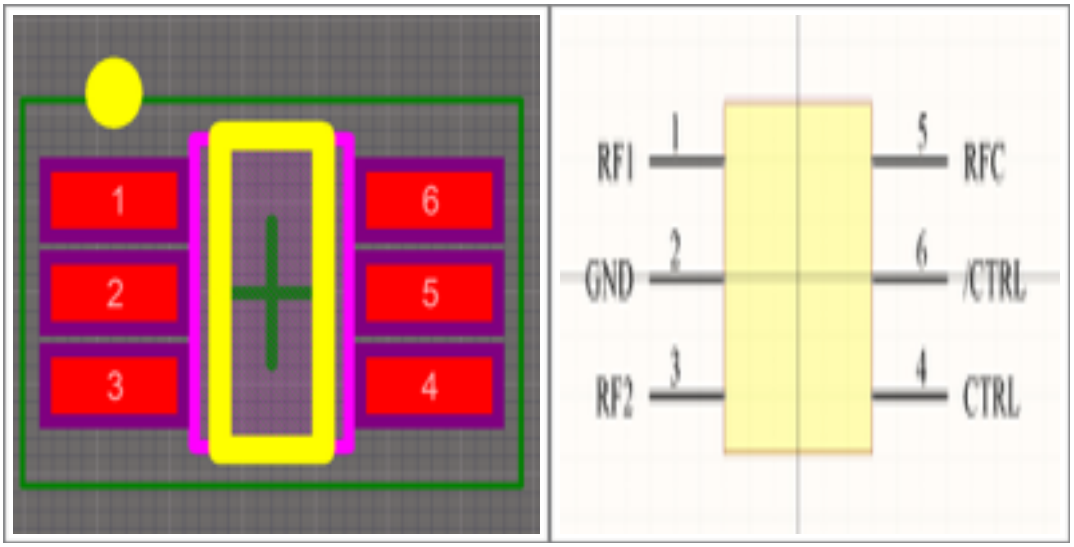


Fig.9 : Schématique et empreinte d'un composant sur Altium(exemple)

Et le routage

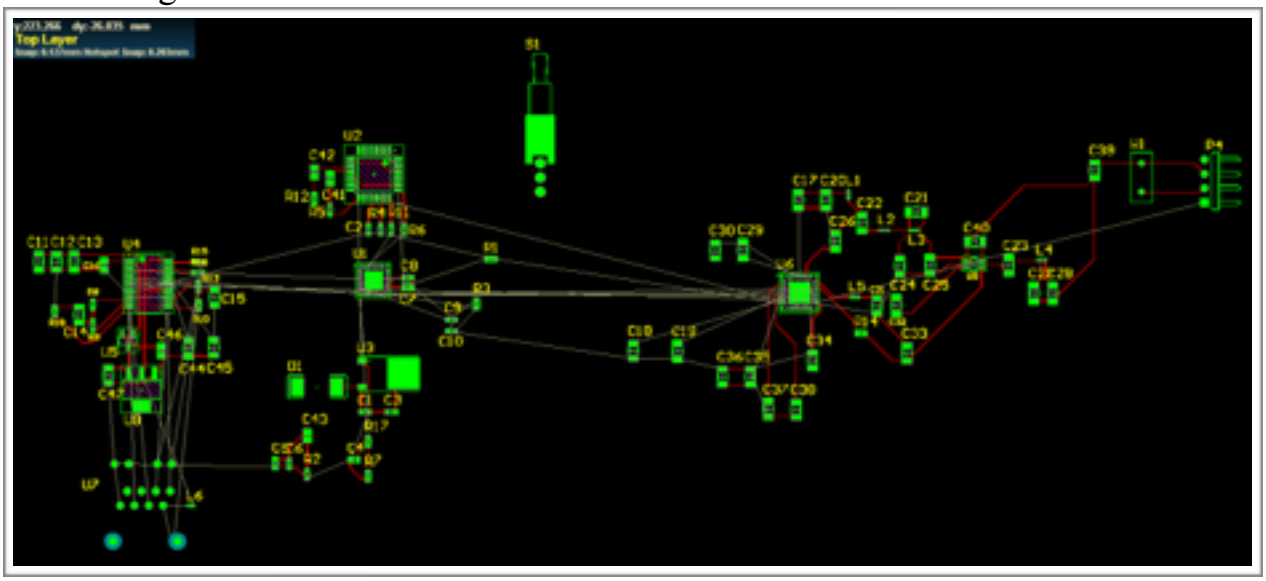


Fig.10 : Routage de la carte sur Altium

- *Logiciel Fritzing*

N’ayant pas eu le temps de finir le routage sur Altium, nous l’avons réalisé, sur un autre logiciel électronique Fritzing (schéma puis routage). Les cartes ont été tirées par un industriel et livrées prêtes pour la soudure des composants.

Une carte a pu être soudée. Malheureusement, nous nous sommes rendus compte, lors de la soudure, que l’empreinte disponible sur Fritzing QFN 28 ne correspondait pas exactement au package du SX1276.

Nous avons quand même réalisé les tests des différents étages. Pour l’instant, l’étage d’alimentation fonctionne correctement. D’ autres étages sont à vérifier : FTDI (convertisseur USB série) , étage Ethernet, étage Arduino,étage SX1276.

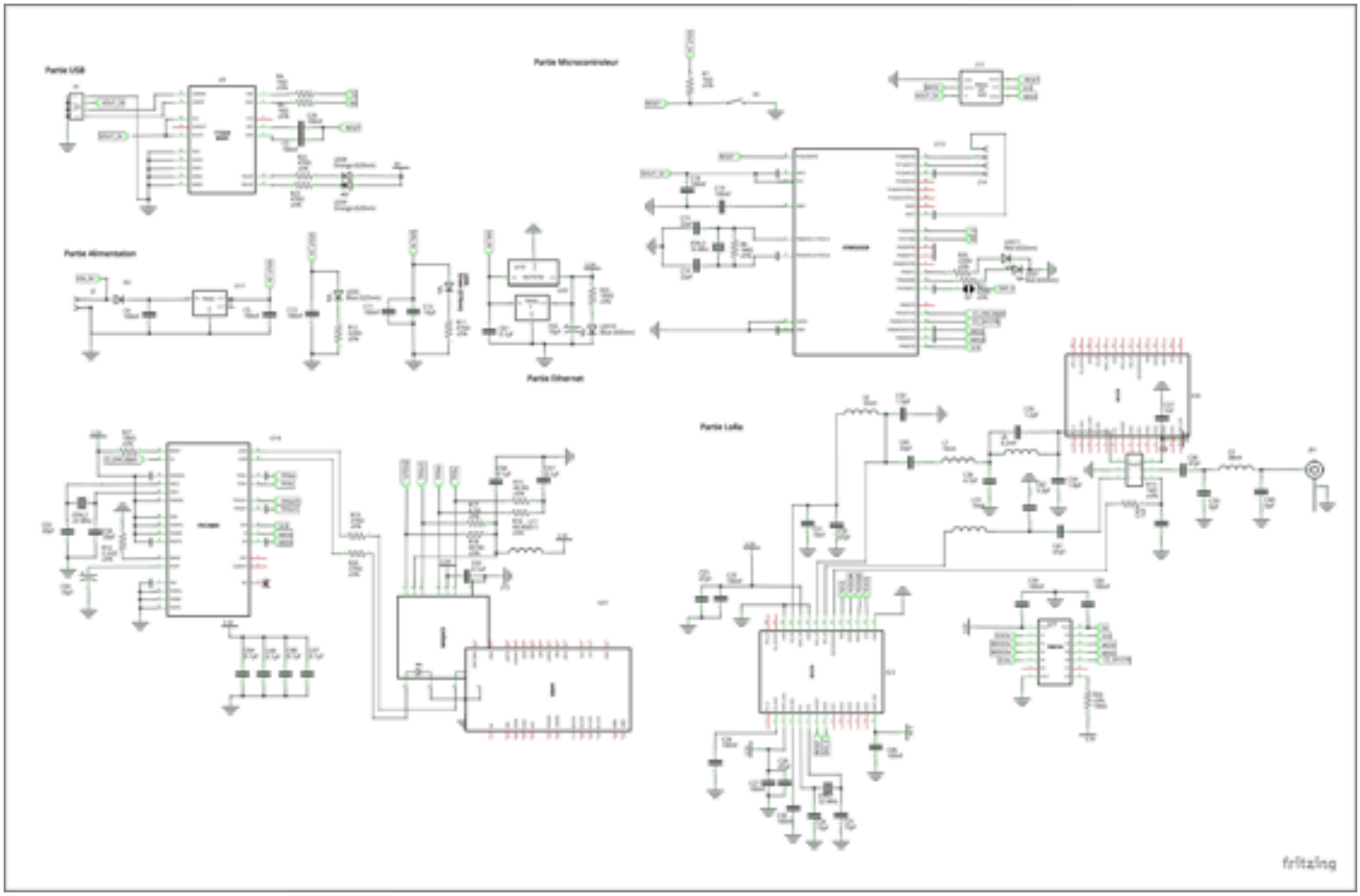


Fig.11: Carte finale design sur Fritzing

2. Difficultés rencontrées

Pendant la réalisation du projet, nous avons rencontré quelques difficultés :

1) Le partage du SPI

Le bus SPI doit être partagé par les 2 périphériques(la carte réseau et le SX1276). L'Arduino ne dispose que d'une seule broche. Il a donc été nécessaire de définir une entrée numérique comme 2 broche Chip Select dans le fichier SPI.c .
Pour pouvoir bien communiquer avec un périphérique, il faut s'assurer que l'autre CS est à l'état haut.

2) La limite de la taille du paquet LoRa

Lorsque le module SX1276 ne peut envoyer que 255 octets, il risque que LoRa n'arrive pas à transmettre tout le paquet que ENC28J60 lui envoie. Donc au lieu d'envoyer un paquet complet, la carte réseau et LoRa vont envoyer des fragments qui contiennent 64 octets du paquet à la fois. (voir annexes)

3) L' adaptation des signaux

Lors du test des modules inAir9, nous nous sommes rendus compte que les signaux n'étaient pas correctement adapté, ce qui avait pour conséquence sur la communication série. Nous avons donc introduit des ponts diviseurs, ce qui a résolu le problème.

Cependant, il existe des adaptateurs de niveau dans le commerce comme le TXB0108 qui a l'avantage d'être bi-directionnel.

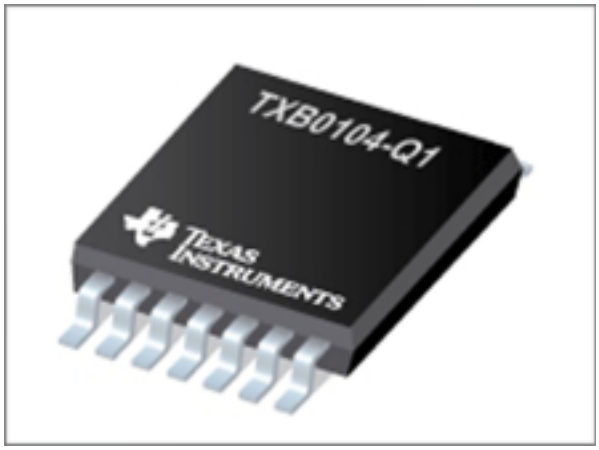


Fig.11: Adaptateur de niveaux

3. Perspectives

Quelques points à développer sont envisageables. En effet, pour pallier au problème cité précédemment sur la communication non simultanée, on pourrait mettre une deuxième antenne de réception(ou émission, full duplex) sur chaque plateforme et ainsi éviter le problème de blocage du micro-contrôleurs. Voici l'idée en image :

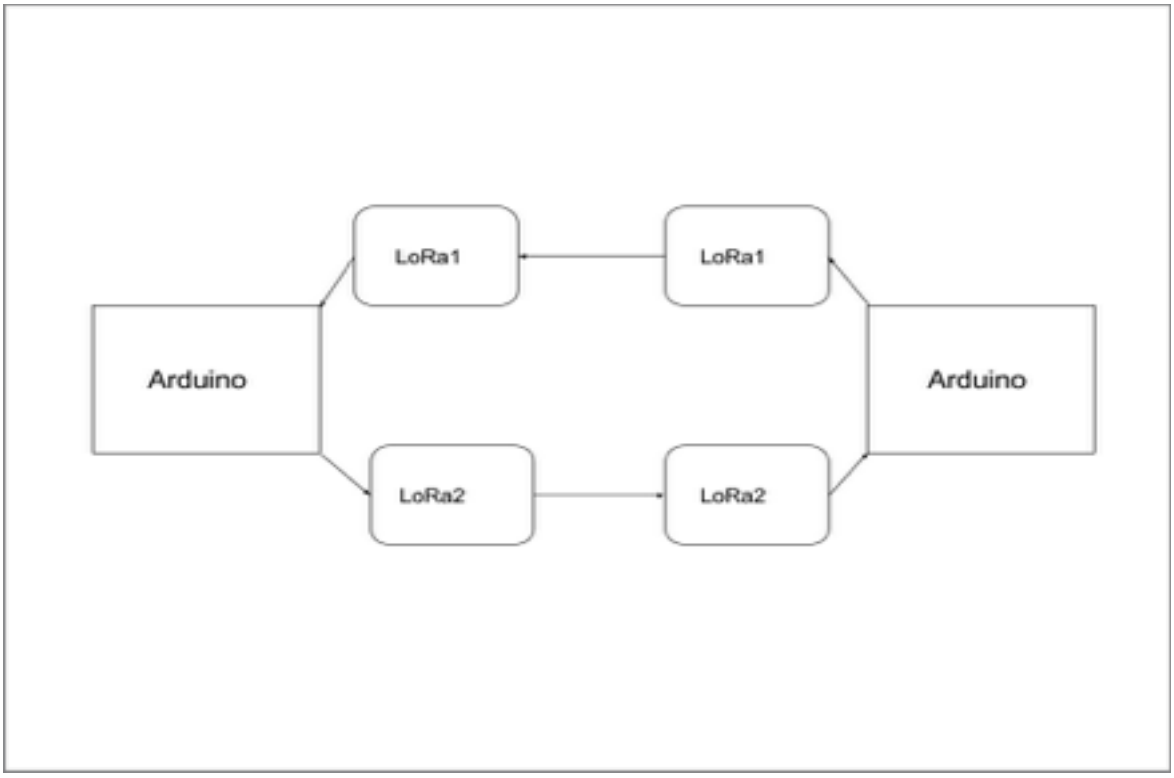


Fig.12: prototype full duplex avec 2 antennes LoRa

- Un autre axe d'amélioration du projet serait de porter le code sur d'autres plateformes autres que AtMega328p, et essayer de le faire fonctionner sur des architectures de type ARM cortex (avec plus de mémoire, vitesse d'exécution,, périphériques, …)

- Un indicateur utile est le RSSI, il enseigne sur la qualité de transmission entre les modules LoRa. Des études plus poussées sur la portée peuvent être effectuées, en tenant compte également du SNR, du RSSI par paquet transmis,..

III. Conclusion

En conclusion, nous pouvons dire que ce projet nous a permis d'améliorer la gestion du travail en équipe et la gestion des problèmes inattendus.

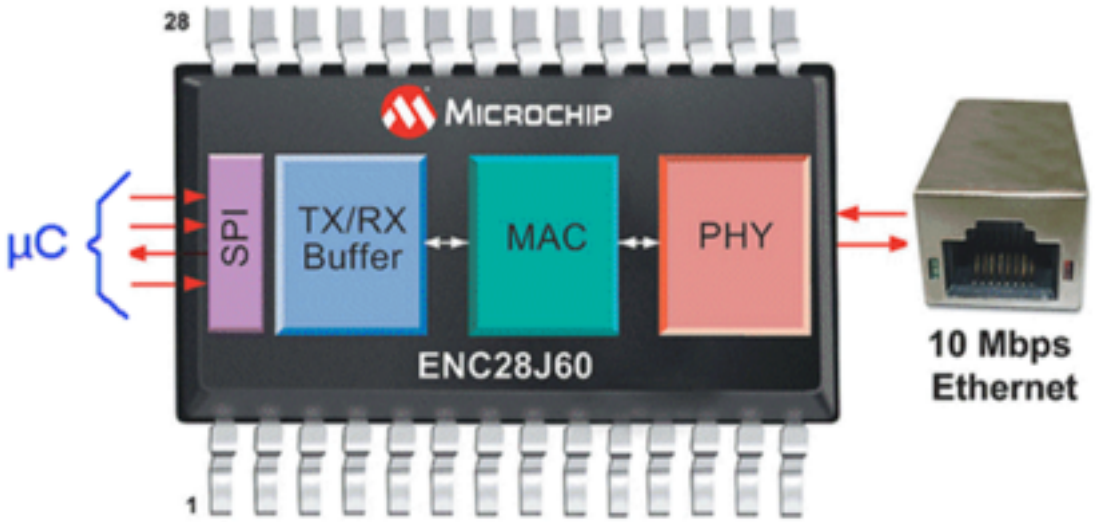
Quant au projet, il est peut être utilisé pour des applications telles l'envoi des mails(format texte)qui ne nécessite que d'un faible débit, échange d'informations, ..

La portée du réseau étant d'environ 15 km en milieu rural (moins en zone urbaine), on pourrait très bien l'améliorer avec un réseau maillé pour couvrir de plus grandes zones.

ANNEXES

- ***Comment fonctionne la carte Ethernet ENC28J60?***

La puce **enc28j60** Microchip présente le premier contrôleur Ethernet 10BASE-T(10Mbits/s) référence ENC28J60 se laissant facilement piloter via un bus SPI(Serial Programming Interface) par un micro-contrôleur et ne comportant que 28 broches pour un boitier au format DIL.
Ce contrôleur Ethernet dispose d'un module MAC et PHY intégrés et de 8ko de mémoire tampon permettant de minimiser la complexité de la mise en oeuvre et le coût.
Les applications visées sont : la VoIP, l'automation industrielle, l'automation de bâtiment, l'instrumentation, la sécurité et la domotique.
Jusqu'à lors, on ne trouvait sur le marché que des contrôleurs Ethernet tels que le RTL8019AS de Realtek. Ce circuit n'existait qu'au format CMS (boitier PQFP) composé d'une centaine de broches dont le tiers étaient destinées au bus de communication parallèle (adresses et données). Bref, sa mise en oeuvre était pour le moins ardue que ce soit sur le plan du routage du circuit imprimé que pour la soudure du composant.
Ainsi, grâce à son faible nombre de broches faciles à souder, le ENC28J60 permet une intégration sensiblement facilitée dans votre réseau domestique en lui couplant à un micro-contrôleur afin de piloter les fonctionnalités Ethernet. Disponible en boitier DIL (DIP), il reste la solution la moins chère pour la mise en oeuvre d'un projet connecté au réseau Ethernet.



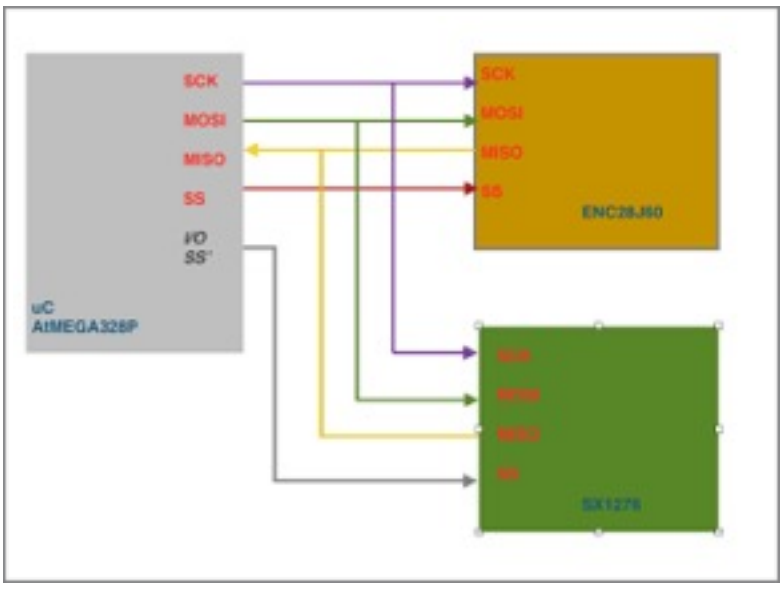
Il y a 7 sections de fonctionnalités pour ENC28J60.
1. Un Interface SPI : un canal de communication entre le contrôleur hôte et l'ENC28J60.
2. Les registres de control : contrôle et surveille l'ENC28J60.
3. Un buffer RAM avec doubles ports : reçoit et transmet des paquets de données.

4. Un arbitre : contrôle l'accès du buffer RAM quand des demandes sont faites de DMA, transmet et reçoit des blocs.
5. Un interface bus : interprète des données et des commandes reçues via l'interface SPI.
6. Un module MAC(Medium Access Control) : implémente IEEE 802.3 qui conforme MAC logique.
7. Un module PHY (Physical Layer) : code et décode des données analogiques qui sont présentées sur l'interface à paire torsadée.

- *Comment fonctionne le module SX1276?*

Le modem LoRa utilise une technique exclusive de modulation du spectre étalé opérant dans la bande de fréquences ISM. Cette technique est plus résistante aux interférences et au bruit dans la bande.
La modulation LoRa repose sur le même principe que la modulation DSSS(Direct Sequence Spread Spectrum) qui consiste donc à générer de la redondance d'information à chaque envoi d'une séquence de bits. Ainsi les bits reçus qui n'utilisent pas le même codage seront rejetés (dont les signaux d'interférence ou le bruit). Le récepteur peut ainsi reconnaître l'émetteur. Dans la modulation LoRa, l'étalement de spectre est obtenu en générant un signal qui varie continuellement en fréquence et occupe toute la bande passante.

Partage du bus SPI



Il existe plusieurs types de communication

- ***Simplex :***

En mode simplex, lorsqu'un canal de communication peut uniquement envoyer des informations dans un sens; par exemple communication par fibre optique,..

- ***Half duplex :***

Half-duplex est utilisé pour décrire la communication où seulement un côté peut parler à la fois. Une fois qu'un côté a fini de transmettre ses données, l'autre côté peut répondre. Si les deux tentent de parler en même temps, une collision peut se produira sur le réseau.

- ***Full duplex :***

Ce mode de communication est utilisé pour décrire la communication où les deux parties sont capables d'envoyer et de recevoir des données simultanément. Dans ce cas, il n'y a pas de risque de collision. C'est le mode qui nous intéresse.

Code principal dans Bridge

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>

#include "serial.h"
#include "SPI.h"
#include "enc28j60.h"
#include "SX1276.h"

#define PACKET_FIN      0x80
#define MAX_FRAGMENT64
#define MAX_WAIT        1000
#define GRAIN_WAIT      10
#define MAX_ETHERNET 1514

unsigned char mac[]={0x00,0x10,0x10,0x10,0x10,0x10,MAC_LOW_BYTE};
unsigned char packet_lora[MAX_FRAGMENT+1];
unsigned char packet_received[MAX_FRAGMENT+1];
int nb_ethernet=0;
unsigned char nb_received=0;
int index_ethernet;
unsigned char num_frag;
static int num=-1;

//control the LEDs
void output_set(int port,unsigned char value)
{
```

```
        if(value==0) PORTD &= ~(1<<port); else PORTD |= (1<<port);
}

void io_init(void)
{
        DDRD |= 0x10;
        DDRD |= 0x20;
        DDRD |= 0x40;
        DDRD &= ~0x80;
}

int main(void)
{
mode      = 0x01;
Freq_Sel          = 0x00;
Power_Sel         = 0x00;
Lora_Rate_Sel  = 0x01;
BandWide_Sel   = 0x08;
Fsk_Rate_Sel   = 0x00;

io_init();
enc28j60IOSetup();
spi_init();
init_printf(9600);
printf("debug start\n");
enc28j60Init(mac);
printf("debug half\n");
sx1276_Config();
printf("debug stop\n");
unsigned char rxmode=0;
while(1)
        {
        if(nb_ethernet==0)
        {
                nb_ethernet = enc28j60PacketReceive();
                if(nb_ethernet>0 && nb_ethernet<=MAX_ETHERNET)
                {
                index_ethernet = 0;
                num_frag = 0;
                output_set(5,1);
                }
                else nb_ethernet=0;
        }
#ifdef VERBOSE
        printf("num=%d nb_ethernet=%d\n",num,nb_ethernet);
        _delay_ms(100);
#endif
        if(num<0)
        {
                if(nb_ethernet>0)
                {
                int i;
                num=-1; rxmode=0;
                if(index_ethernet < nb_ethernet)
                {
                        for(i=0;(i<MAX_FRAGMENT) && (index_ethernet<nb_ethernet);i++)
                        {
                        unsigned char c = enc28j60ByteRead();
                        index_ethernet++;
                        packet_lora[i]=c;
                        }
                        if(index_ethernet < nb_ethernet) packet_lora[i]=0x00;
                        else packet_lora[i]=0x80;
                        packet_lora[i] |= num_frag++;
#ifdef DEBUG
                        printf("ether=%d/%d num_frag=%02x taille=%d\n",index_ethernet,nb_etherne,packet_lora[i],+1);
#endif
                        sx1276_LoRaEntryTx();
                        output_set(4,1);
                        sx1276_LoRaTxPacket(packet_lora,i+1);          //transmission from LoraServer
                        sx1276_LoRaTxReady();
                        output_set(4,0);
                        if(index_ethernet >= nb_ethernet)
                        {
                        nb_ethernet = 0;
                        enc28j60FreeMemory();
                        output_set(5,0);
                        /* Attendre un paquet LoRa un temps raisonnable */
                        unsigned char test=0;
```

```
                                sx1276_LoRaEntryRx();
                                rxmode=1;
                                for(i=0;i<MAX_WAIT/GRAIN_WAIT;i++){
                                test=sx1276_LoRaRxTest();
                                if(test) break;
                                _delay_ms(GRAIN_WAIT);
                                }
                                if(test){
#ifdef DEBUG
                                printf("Got answer!\n");

#endif
                                }
                                }
                                else _delay_ms(10);
                        }
                        }
                        else{
                        if(num==-2){
                                unsigned char c=0;
#ifdef DEBUG
                                printf("Send void packet\n");
#endif
                                sx1276_LoRaEntryTx();
                                sx1276_LoRaTxPacket(&c,1);
                                sx1276_LoRaTxReady();
                                num=-1;
                        }
                        }
                }
                if(index_ethernet<=0 || nb_ethernet==0){
                        if(!rxmode) sx1276_LoRaEntryRx();
                        rxmode=1;
                        }
                if(rxmode && (nb_received=sx1276_LoRaRxPacket(packet_received,MAX_FRAGMENT+1))>0)
                {
                        unsigned char header=packet_received[nb_received-1];
#ifdef DEBUG
                        printf("rec=%d h=%02x\n",nb_received,header);
#endif
                        if (nb_received>1)
                        {
                        int j;
                        if(num>=0 && (header&0x7f)!=num+1) num=-1;
                        else{
                                if(num<0) num=-1;
                                if((header&0x7f)==0x00)
                                {
                                enc28j60InitializeSend();
                                output_set(6,1);
                                }
                                for(j=0;j<nb_received-1;j++) enc28j60WriteByteSend(packet_received[j]);
                                if((header&0x80)==0x00) num++;
                                else {
#ifdef DEBUG
                                printf("Ether sending num=%d\n",num);
#endif
                                num=-2;
                                enc28j60FinalizeSend();
                                output_set(6,0);
                                }
                                }

                        }
                        rxmode=0;
#ifdef DEBUG
                        printf("num=%d\n",num);
#endif
                }
                }
}
```

```
# Makefile pour l'executable du relai
#
#

MCU = atmega328p
TERM_DUE1 = /dev/ttyUSB0
TERM_DUE2 = /dev/ttyUSB1
TERM_UNO1 = /dev/ttyACM0
TERM_UNO2 = /dev/ttyACM1
PGMER_DUE1 = -c arduino -b 57600 -P $(TERM_DUE1)
PGMER_DUE2 = -c arduino -b 57600 -P $(TERM_DUE2)
PGMER_UNO1 = -c stk500v1 -b 115200 -P $(TERM_UNO1)
PGMER_UNO2 = -c stk500v1 -b 115200 -P $(TERM_UNO2)
DUDE = /usr/bin/avrdude -F -v -p $(MCU)

CFLAGS += -I ../Serial -I ../SPI -I ../Enc28J60 -I ../SX1276

LOCAL_LIBS += -L ../Serial -L ../SPI -L ../Enc28J60 -L ../SX1276
LOCAL_LIBS += -lserial -lSPI -lENC28J60 -lSX1276

TARGET = bridge

C_SRC = $(wildcard *.c)
OBJS = $(C_SRC:.c=.o)

all: $(TARGET).hex

clean:
  rm -f *.o *.hex *.elf $(TARGET)

%.o:%.c
  $(CC) -c $(CFLAGS) $< -o $@

$(TARGET).elf: $(OBJS)
  $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LOCAL_LIBS)

$(TARGET).hex: $(TARGET).elf
  avr-objcopy -j .text -j .data -O ihex $(TARGET).elf $(TARGET).hex

reset_due1:
  stty -F $(TERM_DUE1) hupcl # reset

reset_due2:
  stty -F $(TERM_DUE2) hupcl # reset

reset_uno1:
  stty -F $(TERM_UNO1) hupcl # reset

reset_uno2:
  stty -F $(TERM_UNO2) hupcl # reset

upload_due1: reset_due1 $(TARGET).hex
  $(DUDE) $(PGMER_DUE1) -U flash:w:$(TARGET).hex

upload_due2: reset_due2 $(TARGET).hex
  $(DUDE) $(PGMER_DUE2) -U flash:w:$(TARGET).hex

upload_uno1: reset_uno1 $(TARGET).hex
  $(DUDE) $(PGMER_UNO1) -U flash:w:$(TARGET).hex

upload_uno2: reset_uno2 $(TARGET).hex
```

```
	$(DUDE) $(PGMER_UNO2) -U flash:w:$(TARGET).hex

#
# Makefile du PFE Relai Ethernet LoRa
#

#
# Constantes pour la compilation des programmes
#

export MCU = atmega328p
export F_CPU = 16000000
FLAGS = -mmcu=$(MCU)

export CLIB = avr-ar cq
export CC = avr-gcc
export LD = avr-gcc
export CFLAGS = -Wall $(FLAGS) -DF_CPU=$(F_CPU) -Os -DDEBUG
export LDFLAGS = $(FLAGS)

#
# Constantes liees au projet
#

DIRS= SPI Serial Enc28J60 SX1276 Bridge

#
# La cible generale
#

arduino1: CFLAGS += -DMAC_LOW_BYTE=01
arduino1: $(patsubst %, _dir_%, $(DIRS))

arduino2: CFLAGS += -DMAC_LOW_BYTE=02
arduino2: $(patsubst %, _dir_%, $(DIRS))

$(patsubst %,_dir_%,$(DIRS)):
	cd $(patsubst _dir_%,%,$@) && make

_dir_Bridge: _dir_SPI _dir_Serial _dir_Enc28J60 _dir_SX1276

#
# La cible de nettoyage
#

clean: $(patsubst %, _clean_%, $(DIRS))

$(patsubst %,_clean_%,$(DIRS)):
	cd $(patsubst _clean_%,%,$@) && make clean
```

Carte routée sur Fritzing