Rapport du Travail d'Etude et de Réalisation

Communication entre plusieurs Arduino DUE à l'aide du bus CAN à des fins de contrôler un tableau de bord de voiture

Réalisé par Maxime DEREGNIEAUX, Milehana JOSEPH et Lucas LOPEZ

RESUME

Dans le cadre de notre projet de travail d'étude et de réalisation, nous avons eu à réaliser une communication entre plusieurs Arduino DUE à l'aide du bus CAN. L'Arduino DUE a deux contrôleurs CAN intégrés mais n'a pas de système d'émission/réception.
Nous allons donc vous présenter dans un premier temps la solution que nous avons utilisée afin de permettre la communication et par la suite nous vous présenterons le déroulement de notre projet.

Table des Matières

GLOSSAIRE

CAN : Controller Area Network (voir introduction)

Shield Arduino : Les shields sont des cartes que l'on branche sur des cartes microcontrôleur, comme des cartes Arduino,permettant d'ajouter des fonctionnalités à celles-ci tout en préservant les broches d'entrée/sorties.

PCB : C'est un circuit imprimé (Printed Circuit Board en anglais)

Cadsoft Eagle : Logiciel de CAO dédié au routage de PCB. Il permet de créer des modèles de circuits imprimés à partir d'un schéma électrique mais aussi de générer des fichiers gerber.

Fichiers gerber : Fichiers contenant les paramètres de construction d'un PCB (taille de la carte, diamètre des trous de perçage…etc)

INTRODUCTION

Le bus CAN (Controller Area Network) est un protocole de communication série, développé par la société BOSH et l'université de Karlsruhe au début des années 1980. Il supporte des systèmes temps réel avec un haut niveau de fiabilité. L'objectif était de répondre à la problématique du câblage de plus en plus présent dans les automobiles à partir des années 1960 (de l'ordre de 2 km par véhicule). De ce fait, la fiabilité et la sécurité dans les voitures étaient menacées.

Dans un premier temps destiné à l'industrie automobile, le bus CAN pouvait fournir une solution de multiplexage des informations circulant à bord de la voiture à faibles coûts. Le nombre de câbles a donc diminué, et dans le même temps, le nombre calculateurs et capteurs distribués dans le véhicule a augmenté.

Avec le bus CAN, les stations ayant les mêmes droits (organes de commande, capteurs ou actionneurs) sont reliées par un bus série. Le protocole CAN, ainsi que les paramètres électriques de la ligne de transmission, sont fixés par la norme 11898. La transmission physique s'effectue soit par une paire torsadée ou par liaison infrarouge, hertzienne ou par fibre optique.

Ce document présentera un exemple d'implémentation d'un bus CAN. Pour cela, nous disposons de 3 cartes ARDUINO DUE, de 2 maquettes de tableau de bord de voitures pour la réalisation d'un test d'utilisation. Enfin, il faut savoir que nous avons dû ajouter un circuit d'adaptation série vers différentiel car c'est ainsi que fonctionne le bus CAN. Les détails de ce circuit d'adaptation et l'implémentation logicielle seront développés dans la suite du document.

PARTIE MATERIELLE

I. ROUTAGE

1) Choix de l'électronique

Comme mentionné précédemment, l'Arduino Due n'a pas d'émetteur-récepteur intégré (transceiver en anglais). Il a donc fallu se procurer un composant supplémentaire. Il existe plusieurs façons de parer à ce problème. Nous nous sommes concentrés sur une seule en particulier, le composant émetteur-récepteur MCP2551 car il est très souvent utilisé, peu cher et donc simple à se procurer. Il est aussi facile à manipuler du fait de sa taille (composant à broches). Un autre choix possible était le SN65HVD234 mais celui-ci est un CMS (Composant monté en surface) et donc plus difficile à implémenter.

Une fois le composant choisi, nous nous sommes basés sur le schéma suivant pour réaliser le montage:

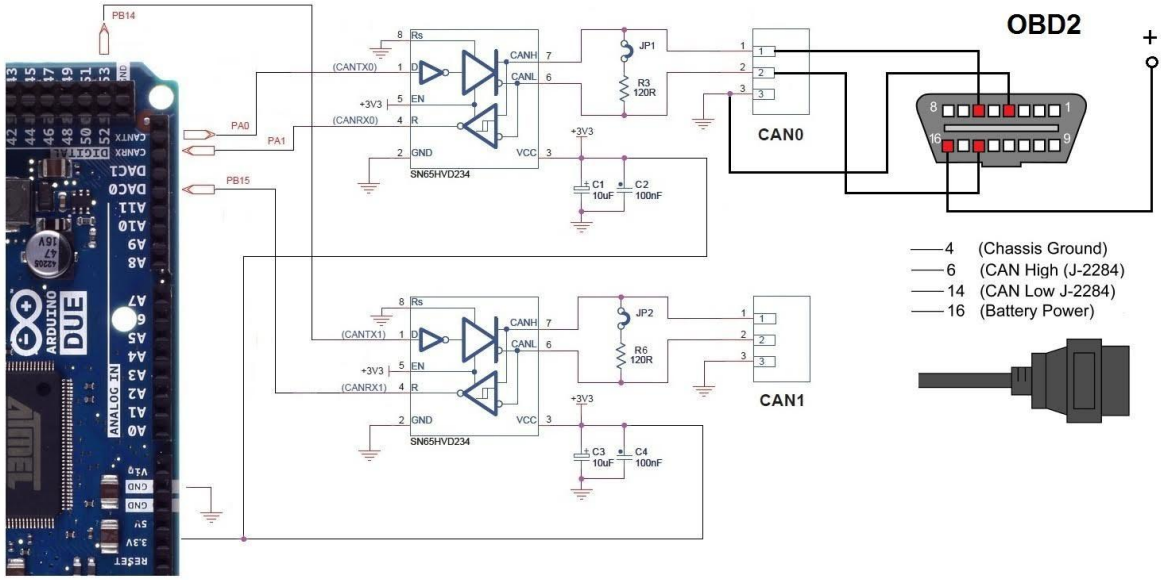

Figure 1: Schéma d'appui du circuit d'adaptation

Dans ce montage, l'auteur a décidé d'utiliser l'émetteur-récepteur SN65HVD234 sur les deux CAN de l'Arduino Due, ça lui permet de gérer plus choses mais nous n'avons pas besoin des deux. Dans notre projet nous ne nous intéressons qu'à la partie supérieure. Ainsi, chaque Arduino sera doté d'un seul transceiver capable d'émettre et de recevoir sur le bus CAN.

Les couples de condensateur sont utilisés pour découpler l'alimentation et éviter des phénomènes parasites tels que de légères chutes de tension et des composantes alternatives non voulues.

La résistance de 120 Ω est la résistance de terminaison des bus CAN. Cette resistance permet d'éviter les phénomènes de réflexion d'onde et limite donc les interférences. Dans le secteur automobile on utilise la norme 11898-2 qui utilise aussi des câbles d'impédance 120 Ω, on obtient ainsi une bonne adaptation d'impédance.

Il a tout de même fallu adapter le montage ci-dessus puisque notre transceiver nécessite une tension d'alimentation de 5V et non 3.3V. Par conséquent, la tension de sortie du MCP2551 est un signal variant de 0V à 5V. Il a donc été nécessaire de protéger l'Arduino des surtensions en abaissant cette tension de sortie à 3.3V. Pour cela nous avons réalisé un pont diviseur de tension.

Le schéma final que nous avons réalisé est le suivant :

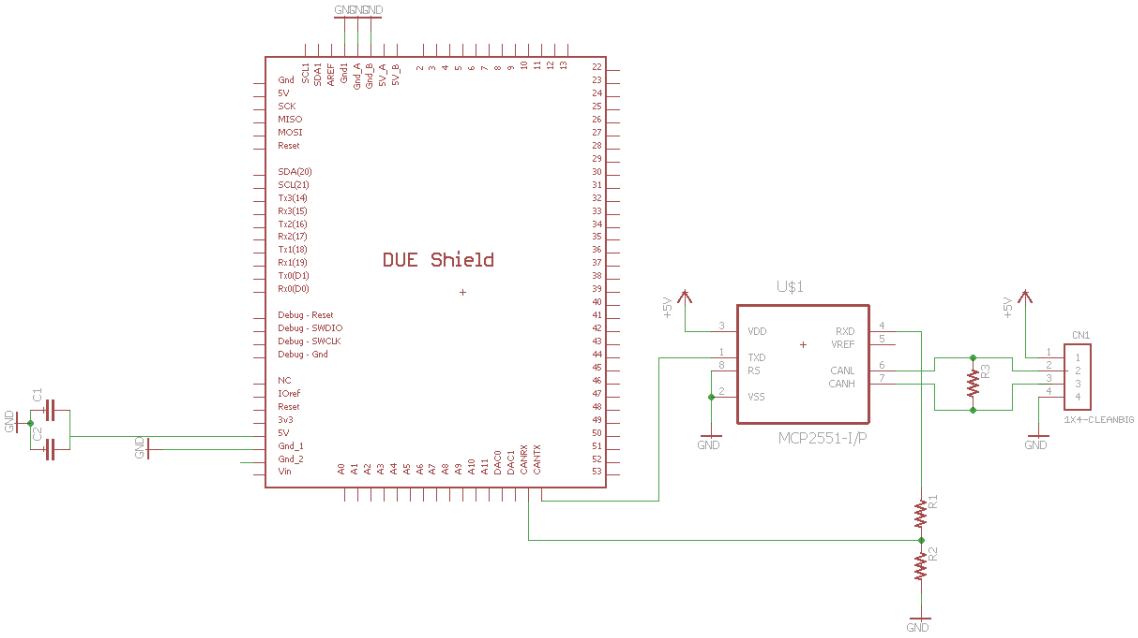

Figure 2: Schéma du circuit d'adaptation final

Comme nous pouvons le voir, on retrouve les condensateurs de découplage $C_1 = 10\mu F$ et $C_2 = 100nF$. Ces condensateurs permettent de supprimer les parasites de certaines fréquences en choisissant leur valeur.
On retrouve aussi la résistance de terminaison $R_3 = 120\Omega$ du schéma précédent.
Pour le pont diviseur de tension, il nous faut calculer les résistances
$V_{arduino} = \frac{R_2}{R_2+R_1} * V_{MCP2551}$ du coup $\frac{V_{arduino}}{V_{MCP2551}} = \frac{3,3}{5} = 0,66V = \frac{R_2}{R_2+R_1}$ $\Rightarrow$ $R_2 = 1,95 * R_1$
Nous avons donc choisi des résistances qui respectent cette égalité.

Ci-dessous, vous trouverez un récapitulatif des valeurs de chaque composant :

C1	10 µF
C2	100 nF
R1	3,6 kΩ
R2	6,8 kΩ
R3	120 Ω

Tableau 1: Tableau récapitulatif du schéma d'adaptation

2) Conception assistée par ordinateur : CadSoft Eagle

Pour réaliser le routage de notre shield arduino nous avons utilisé la version gratuite du logiciel « Eagle » de *CadSoft*.*
Nous avons commencé par dessiner notre schéma Figure 2 en faisant attention de choisir des composants de taille appropriée. Ensuite, nous sommes passés en mode « board » pour faire le routage.

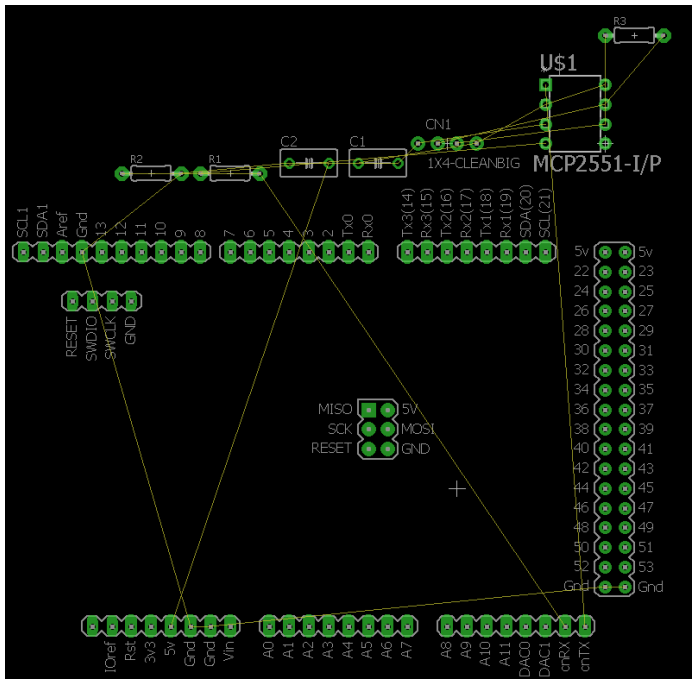

Figure 3 : Première étape du routage

Dans ce mode nous avons accès aux différents points de contact des composants. Nous avons placé les composants intelligemment pour que l'on puisse les relier sans problèmes. Après avoir effectué le routage, on obtient la carte suivante :

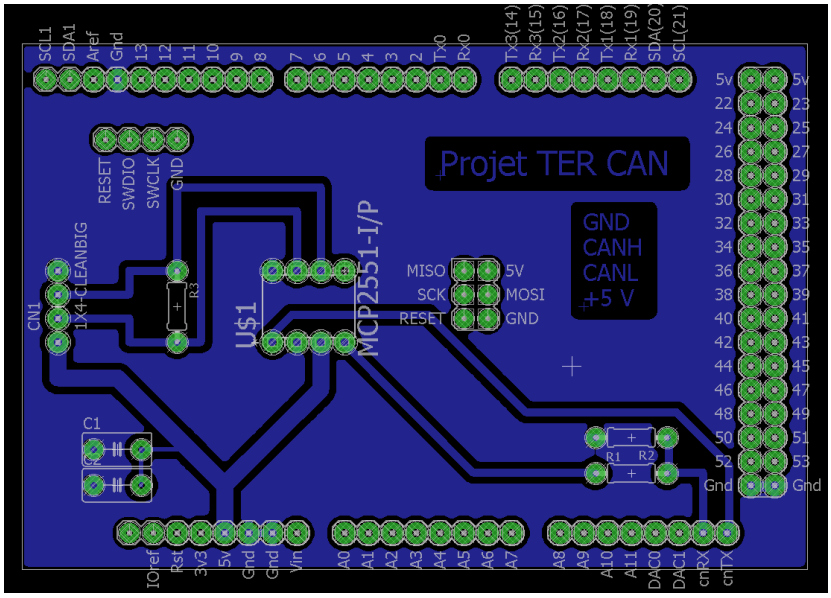

Figure 4: Seconde étape du routage

Sur l'image ci-dessus, les parties bleues correspondent au cuivre du PCB*, les noires correspondent à l'isolant et les vertes aux contacts des composants.
La dernière étape était de créer un fichier gerber* définissant les paramètres de construction de la carte, tel que la taille du perçage et le chemin de découpe. Ce fichier est utilisé par les fabricants pour créer le PCB*.

II. PROTOTYPE DU SHIELD ARDUINO DUE

1) Fabrication de la carte

Pour un gain de temps et par envie de découvrir, nous avons créé le circuit imprimé nous-même en utilisant de l'encre permanente pour dessiner les pistes et un mélange d'acide chloridrique et d'eau oxygénée pour nettoyer la carte et ne laisser que les pistes.
Puisque ce processus n'est pas pertinent au fonctionnement du bus CAN, il ne sera pas décrit dans ce rapport.
Voici le résultat de notre travail :

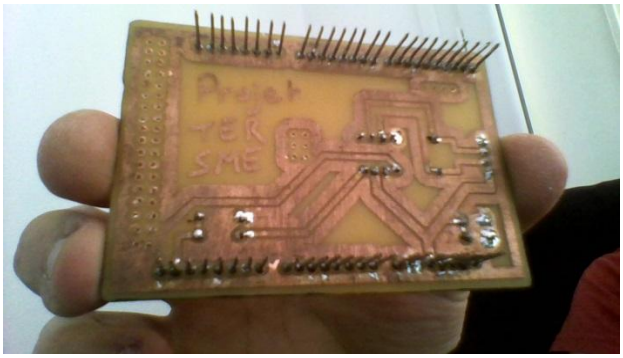

Figure 5: Partie cuivrée de la carte avec les pistes

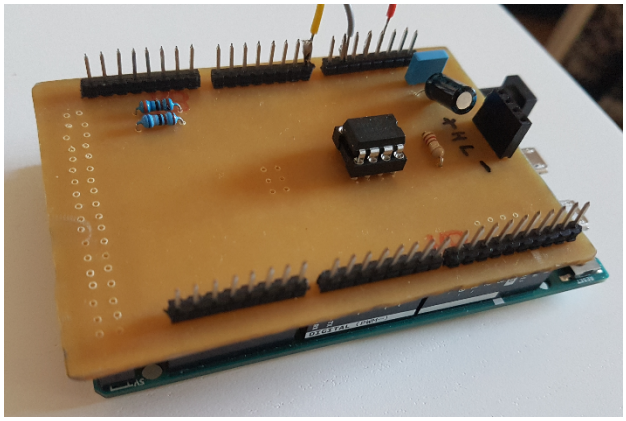

Figure 6: Partie composant de la carte

Comme vous le voyez sur la **Figure 6**, la carte s'emboite bien sur l'Arduino donc les dimensions sont bonnes.

2) Essai du fonctionnement

Pour tester la continuité des pistes et vérifier qu'il n'y avait pas de faux contacts, nous avons utilisé un multimètre en mode continuité (ohmmètre). La continuité des pistes ayant été vérifiée, nous avons pu procéder aux essais avec l'Arduino.

PARTIE LOGICIELLE

I. Choix de la librairie

Nous avons décidé d'utilisé la librairie "due_can", une librairie open-source, créée par des utilisateurs d'Arduino. Cette librairie est disponible sur GitHub (https://github.com/collin80/due_can), un site de partage de documents privé ou public. Cette librairie est la plus utilisée car elle est très complète. En effet, on peut presque tout paramétrer : Les données, les priorités, les différentes longueurs de trames…

La librairie inclue aussi un fichier texte « HowToUse » composé de toutes les fonctions de la librairie et de leurs explications en anglais. Cela nous a permis de la comprendre un peu mieux et de pouvoir l'utiliser correctement.

II. Utilisation des différentes fonctions

Initialisation :

Pour commencer, il faut initialiser le port du bus CAN. La librairie contient la fonction begin() qui permet d'initialiser le bus à une vitesse de fonctionnement choisie qui est en baud par seconde.
Voici un exemple d'utilisation :

```
Can0.begin(CAN_BPS_250K);
```

Ici, on initialise le port Can0 à la vitesse de 250 KBaud/s.

Paramétrage des trames :

Pour préparer les trames à envoyer, on utilise le type CAN_FRAME décrit ci-dessous et tiré du fichier texte « HowToUse »:

```
typedef struct
{
  uint32_t id;    // Can be either 11 or 29 bit ID
  uint32_t fid;       // Family ID is a somewhat advanced thing. You can ignore it.
  uint8_t rtr;    // Remote Transmission Request - Don't use this.
  uint8_t priority; // Priority for TX frames. Probably not that useful normally
  uint8_t extended;//Set to true for extended frames, false for standard
  uint8_t length; // Number of data bytes
  BytesUnion data;  // Data bytes
} CAN_FRAME;
```

Figure 7: Définition du type CAN_FRAME

Les paramètres importants sont : id, priority, length et data. Nous ne nous servirons pas des autres paramètres dans ce projet.
Voici un exemple de préparation de trame d'envoi :

```
CAN_FRAME outgoing;
outgoing.id = 0x04;
outgoing.length = 2;
outgoing.data.bytes[0] = 0x01;
outgoing.data.bytes[1] = 0x05;
```

Figure 8: Préparation de la trame d'envoi

Ici, on paramètre l'identification de la trame à 4 grâce l'**id**, on limite le nombre d'octet de donné à 2 avec **length** et on attribue aussi la valeur 1 à l'octet 0 et la valeur 5 à l'octet 1 grâce à **bytes[x]**.

Envoi des trames :
L'envoi de trame se fait simplement avec la fonction sendFrame(trame) :

```
Can0.sendFrame(outgoing);
```

Figure 9: Envoi de trame

Cette ligne de code permet donc d'envoyer la trame précédemment paramétrée sur le bus.

Réception des trames :
Pour recevoir une trame, il faut créer tout d'abord une seconde structure CAN_FRAME pour stocker les données reçues. On utilise ensuite la fonction available() pour détecter si le buffer du bus est vide ou si une trame attend d'être lue. Enfin, on utilise la fonction read() pour lire les données :

```
CAN_FRAME incoming;
if (Can0.available() > 0)
  Can0.read(incoming);
```

Figure 10: Réception d'une trame de données

Il suffit ensuite de lire les variables de la structure incoming pour récupérer les données. Voici un exemple pour récupérer le premier octet de données :

```
int donnee_recu_1 = incoming.data.bytes[0];
```

Figure 11: Lecture des données reçues

Gestion de plusieurs Arduino sur le bus :
Pour la gestion simultanée de plusieurs Arduino nous avons attribué une adresse virtuelle à chaque Arduino récepteur.
Quand on paramètre une trame pour l'envoi, nous spécifions dans le paramètre "id" l'adresse de l'Arduino cible.
Ainsi, il suffit d'ajouter une ligne dans le code du récepteur afin qu'il ne laisse passer que les trames avec le bon "id". Pour le faire on utilise la fonction watchFor():

```
Can0.watchFor(0x07);
```

Figure 12: Fonction d'identification

Cette ligne de code permet de ne voir que les programmes ayant l'id 7 et donc ne laisse pas le reste passer.

III. Simulation d'un tableau de bord de voiture

Pour illustrer le fonctionnement de notre bus CAN, nous avons réalisé deux maquettes simulant des tableaux de bord de voitures. Chaque tableau de bord est relié à un Arduino contrôlant ses différents voyants (phares, klaxon, clignotants) et le compteur de vitesse. Un troisième Arduino, que nous utilisons en tant que maître pour cette application, permet d'envoyer les ordres d'allumage ou d'extinction des différents voyants. Les Arduino sont reliés entre eux par le bus CAN uniquement.

Le schéma de câblage global est donné ci-dessous en plusieurs morceaux :

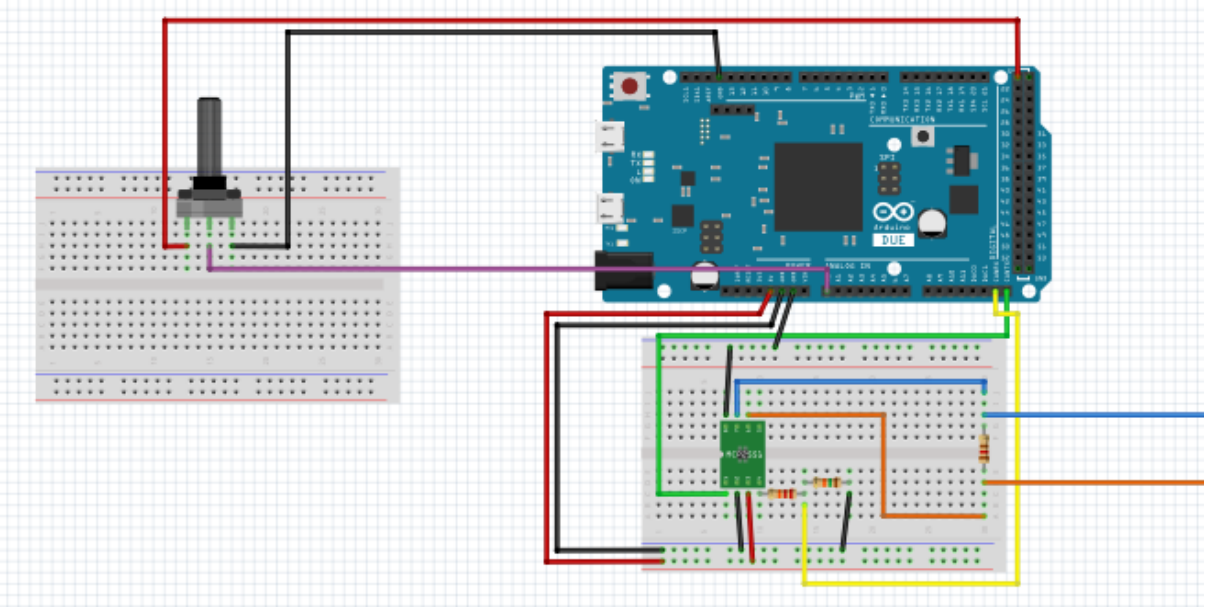

Figure 13: Schéma électrique de l'émetteur

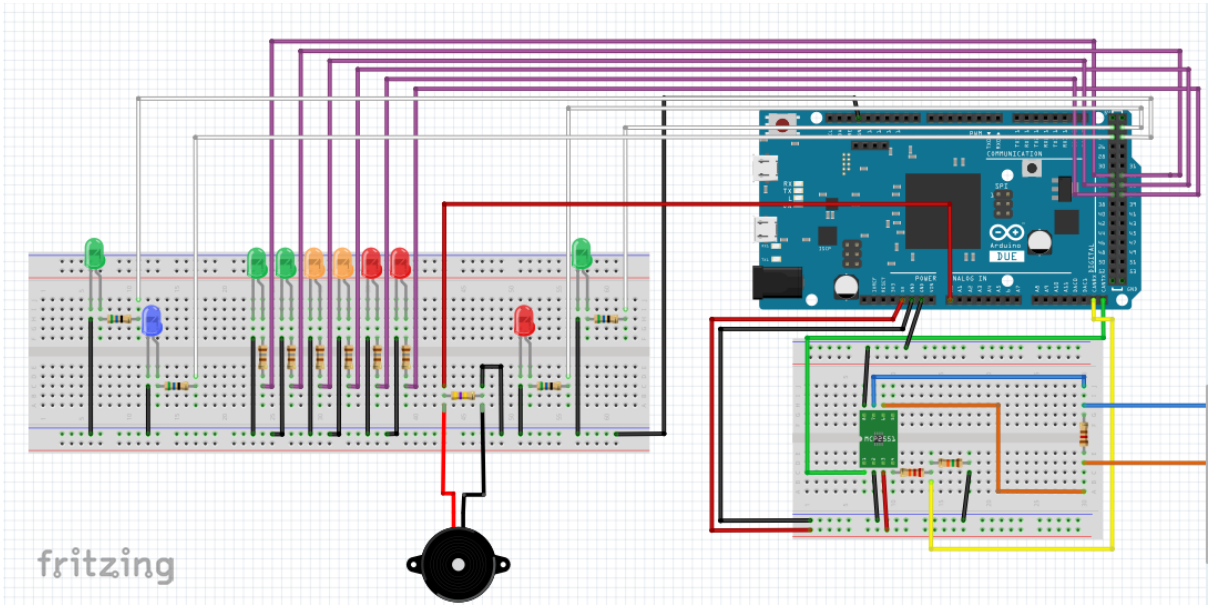

Figure 14: Schéma électrique d'un récepteur

Le second récepteur à exactement la même configuration que celui sur la figure 14. Et les trois montages sont reliés avec les câbles bleus et orange que vous voyez sur la droite des figures 13 et 14.

CONCLUSION

Ce projet a été très complet, car il regroupait à la fois une partie hardware et une partie software. L'alliance des deux rend le projet plus intéressant et les différentes applications à mettre en œuvre par la suite sont diverses et variées.
Il serait intéressant de pouvoir le continuer et l'améliorer. Nous avons vu qu'il pouvait y avoir de la gestion de priorités donc des communications plus nombreuses, dans les deux sens, et ayant des priorités plus ou moins importantes.

Annexes

I. Code de l'émetteur

```
#include <due_can.h>
#include <string.h>
#include "variant.h"

//#define Serial SerialUSB                    //Commande si jamais on utilise l'autre port de
connection de l'arduino

int ValeurTranscript(String mess){            //Retranscription de la commande d'entrée
 int val_retour;
 if(mess == "D"){
  Serial.println("Le clignotant Droit");
  val_retour = 1;
 }
 else if(mess == "L"){
  Serial.println("Allume la lumiere ");
  val_retour = 2;
 }
 else if(mess == "G"){
  Serial.println("Le clignotant Gauche)");
  val_retour = 3;
 }
 else if(mess == "E"){
  Serial.println("Eteind le lumiere");
  val_retour = 4;
 }
 else if(mess == "K"){
  Serial.println("Allume le klaxon");
  val_retour = 5;
 }
 else{
  val_retour = 0;
 }
 return val_retour;
}

int ValeurCompteur( int val){              //Retranscription de la valeur du potar
 int val_retour;
 if(val >= 0 && val < 179){
    val_retour = 1;
 }
 else if(val >= 180 && val < 359){
    val_retour = 2;
 }
 else if(val >= 360 && val < 599){
    val_retour = 3;
```

```
  }
  else if(val >= 600 && val < 799){
     val_retour = 4;
  }
  else if(val >= 800 && val < 999){
     val_retour = 5;
  }
  else if(val >= 1000 && val <= 1023){
     val_retour = 6;
  }
 return val_retour;
}

void setup(){
  Serial.begin(9600);                    //Initialisation du moniteur
  Can0.begin(CAN_BPS_250K);                    //Initialisation du CAN
  Can0.watchFor();                       //Autorise le contact avec tout le monde
  pinMode(A0, INPUT);                       //Déclaration du potar
}

void loop(){
  int potar;
  uint32_t ident;
  CAN_FRAME message;
  String CAN_MSG_1;

  message.length = 8;                      //Initialisation de la taille de la trame

  potar = analogRead(A0);                       //Lecture du potar

  CAN_MSG_1 = Serial.readString();                 //Lecture du moniteur

  if (CAN_MSG_1 == "5")                       //Cas de identifiant
     message.id = 0x05;                    //
  else if (CAN_MSG_1 == "7")                       //
     message.id = 0x07;                    //
  else if (CAN_MSG_1 == "1")                       //
     message.id = 0x01;                    //Permet de choisir avec quel recepteur on veut parler

  message.data.high = 0x0000;                       //On vide la trame
  message.data.low = 0x0000;                        //pour pas créer de mauvaises données
  message.data.bytes[0] = ValeurTranscript(CAN_MSG_1);   //Retranscription de la commande
  message.data.bytes[1] = ValeurCompteur(potar);        //Retranscription de la valeur du
potentiomètre
  CAN.sendFrame(message);                       //Envoi de la trame

}
```

II. Code des récepteurs

```
#include "variant.h"
#include <due_can.h>

//Leave defined if you use native port, comment if using programming port
//#define Serial SerialUSB

void InitTableauDeBord(void){              //Fonction d'initialisation du tableau de bord
 int i = 0;
 do{
   digitalWrite(32, HIGH);
   digitalWrite(22, HIGH);
   digitalWrite(8, HIGH);
   digitalWrite(24, HIGH);
   digitalWrite(23, LOW);
   digitalWrite(35, LOW);
   delay(300);
   digitalWrite(33, HIGH);
   digitalWrite(23, HIGH);
   digitalWrite(22, LOW);
   digitalWrite(36, LOW);
   delay(300);
   digitalWrite(34, HIGH);
   digitalWrite(22, HIGH);
   digitalWrite(23, LOW);
   digitalWrite(37, LOW);
   delay(300);
   digitalWrite(32, LOW);
   digitalWrite(23, HIGH);
   digitalWrite(22, LOW);
   digitalWrite(35, HIGH);
   delay(300);
   digitalWrite(33, LOW);
   digitalWrite(23, LOW);
   digitalWrite(22, HIGH);
   digitalWrite(36, HIGH);
   delay(300);
   digitalWrite(34, LOW);
   digitalWrite(23, HIGH);
   digitalWrite(22, LOW);
   digitalWrite(37, HIGH);
   digitalWrite(8, LOW);
   digitalWrite(24, LOW);
   delay(300);
   i++;
 }while(i<4);
   digitalWrite(34, LOW);
   digitalWrite(23, LOW);
   digitalWrite(22, LOW);
   digitalWrite(37, LOW);
```

```
  digitalWrite(8, LOW);
  digitalWrite(24, LOW);
  digitalWrite(35, LOW);
  digitalWrite(36, LOW);
  digitalWrite(37, LOW);
}

void lecture(){                                    //Fonction de lecture du bus
 int donnee;
 int compteur;
 int cpt = 0;
 CAN_FRAME incoming;

 if (Can0.available() > 0){                        //Test du buffer du bus
   Can0.read(incoming);                            //Lecture des données
   donnee = incoming.data.bytes[0];                //Recupération du premier octet
   compteur = incoming.data.bytes[1];              //Récupération du deuxième octet
 }
 if(incoming.id == 0x05 || incoming.id == 0x01){   //Test de l'id
 switch(donnee){                                   //Traitement des données
   case 1: do{                                     //Clignotant Droit
            digitalWrite(22, HIGH);
            delay(300);
            digitalWrite(22, LOW);
            delay(300);
            cpt++;
          }while(cpt<5);
          break;

   case 2: digitalWrite(24, HIGH);                 //Allume les codes
          break;

   case 3: do{
            digitalWrite(23, HIGH);
            delay(300);
            digitalWrite(23, LOW);
            delay(300);
            cpt++;
          }while(cpt<5);
          break;

   case 4: digitalWrite(24, LOW);     //Eteind les codes
          break;

   case 5: analogWrite(9, 100);       //Allume le klaxon
          digitalWrite(8, HIGH);
          delay(1000);
          analogWrite(9, 0);
          digitalWrite(8, LOW);
          break;

    default: cpt = 0;
```

```
}

switch(compteur){                       //Traitement des données du potentiomètre

   case 1: digitalWrite(32,HIGH);
         digitalWrite(33,LOW);
         digitalWrite(34,LOW);
         digitalWrite(35,LOW);
         digitalWrite(36,LOW);
         digitalWrite(37,LOW);
         break;

   case 2: digitalWrite(32,HIGH);
         digitalWrite(33,HIGH);
         digitalWrite(34,LOW);
         digitalWrite(35,LOW);
         digitalWrite(36,LOW);
         digitalWrite(37,LOW);
         break;

   case 3: digitalWrite(32,HIGH);
         digitalWrite(33,HIGH);
         digitalWrite(34,HIGH);
         digitalWrite(35,LOW);
         digitalWrite(36,LOW);
         digitalWrite(37,LOW);
         break;

   case 4: digitalWrite(32,HIGH);
         digitalWrite(33,HIGH);
         digitalWrite(34,HIGH);
         digitalWrite(35,HIGH);
         digitalWrite(36,LOW);
         digitalWrite(37,LOW);
         break;

   case 5: digitalWrite(32,HIGH);
         digitalWrite(33,HIGH);
         digitalWrite(34,HIGH);
         digitalWrite(35,HIGH);
         digitalWrite(36,HIGH);
         digitalWrite(37,LOW);
          break;

   case 6: digitalWrite(32,HIGH);
         digitalWrite(33,HIGH);
         digitalWrite(34,HIGH);
         digitalWrite(35,HIGH);
         digitalWrite(36,HIGH);
         digitalWrite(37,HIGH);
          break;
```

```
    default: cpt = 0;
    }
 }
}

void setup()
{

  Serial.begin(9600);                    //Initialisation du moniteur
  Serial.println("Debut");

  pinMode(22, OUTPUT);
  pinMode(23, OUTPUT);
  pinMode(24, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(32, OUTPUT);
  pinMode(33, OUTPUT);
  pinMode(34, OUTPUT);
  pinMode(35, OUTPUT);
  pinMode(36, OUTPUT);
  pinMode(37, OUTPUT);


  Can0.begin(CAN_BPS_250K);                          //Initialisation du bus can

  Serial.println("Attend qu'on le choisisse");
  InitTableauDeBord();
}

void loop() {
   lecture();
   delay(100);

}
```