TP BUS CAN ARDUINO

NOM :	Date :

Objectif final :
Mettre en service le bus CAN pour échanger des données entre 2 cartes Arduino Uno équipées de shield bus CAN.

Compétences abordées :

Réaliser	C4.3 : Analyser la structure logicielle. Procéder aux modifications logicielles C4.5 : Tester et valider un matériel
Installer	C5.2 : Exécuter des mesures et tests appropriés.

Savoirs abordés :

Savoir	**Description**
S7.6. Réseaux locaux industriels (RU)	Bus CAN
S8.1. Instruments de mesure	Analyseur logique Analyseur de signaux

Moyens :
- 2 cartes Arduino Uno
- 2 shield bus CAN Seeedstudio
- Fils de liaison mâle/mâle.
- Câble Ethernet de 80 m.
- Analyseur logique Saleae ou Logic Port.

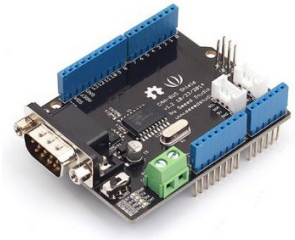
Conditions :
- Analyse et programmation : travail en binôme.
- Mise en œuvre : 2 binômes associés (1 binôme pour l'émission, l'autre pour la réception).
- Les documents nécessaires à la mise en œuvre de ce TP se trouvent sur le site de la section BTS SN.
- Durée : 2H (2 x 1H).
- Compte rendu à la fin de la séance.

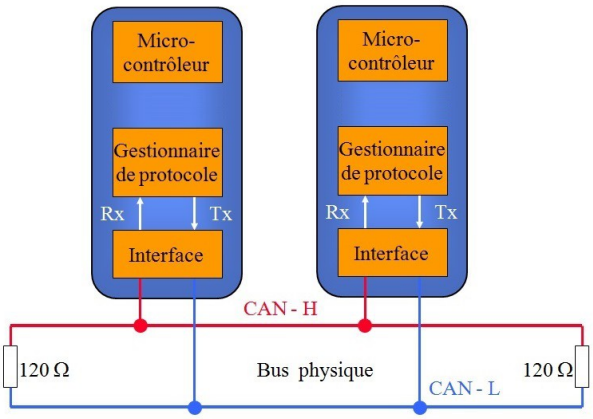
Prérequis :
- Cours sur le bus CAN.

Transmission de données entre 2 Arduino Uno par bus CAN

I. Analyse du shield bus CAN

Les documents des différents composants du shield bus CAN Seeedstudio et de la carte Arduino Uno figurent sur le site.





La représentation en couches de l'association carte Arduino + shield Bus CAN figure ci-dessus.

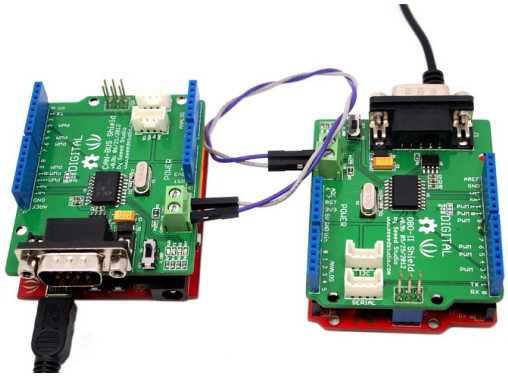
1. Tracer le contour de chaque carte Arduino et de chaque shield bus CAN.
2. Indiquer la référence des composants utilisés pour chacun des éléments de cette représentation en couches.
3. Ajouter si nécessaire un bus de communication intervenant dans les échanges.
4. Indiquer comment il est possible sur les shields de mettre mettre en/hors services les résistances de terminaison.

→

II. Mise en œuvre de la communication entre 2 cartes Arduino

- La mise en œuvre qui suit, en particulier l'installation de la librairie, sera à effectuer en mode administrateur.
- 2 PC se trouvant à proximité l'un de l'autre seront à utiliser, un pour chaque binôme.

5. Prendre connaissance des informations figurant sur le site dédié à la mise en oeuvre du shield seeedstudio *(lien sur le site)*.
6. Télécharger et installer la librairie « CAN BUS Shield ».
7. Mettre en œuvre la communication en liaison avec l'autre binôme en utilisant la librairie. L'une des cartes sera programmée avec le projet « send », l'autre avec le projet « receive_check ».
8. Vérifier que la communication s'effectue bien.
9. Visualiser la trame avec un analyseur logique (Référence de tension sur CANL, signal relevé et interprété par l'analyseur sur CANH. Il est intéressant d'utiliser la voie 0 et de visualiser le signal simultanément en



analogique et en logique). Vérifier la conformité, par rapport au programme, de la valeur visualisée des champs suivants : identificateur, contrôle.

Attention sur une capture Saleae chaque point représente un bit, chaque croix rouge représente un bit-stuffing.

Champ identificateur : → Champ contrôle : → Champ de données : →

Différence entre chronogramme transmis et interprété →

Faire constater

III. Modification du programme et analyse du CRC

10. Modifier les programmes pour que l'identificateur soit 84. Vérifier le bon fonctionnement.
11. Modifier les programmes pour que l'identificateur soit 3 et que la seule donnée transmise soit «0x5B », ainsi la trame correspondra à celle analysée en TD. Relever le CRC *(Cyclic Redundancy Code)* dans la trame. Utiliser les documents illustrant le calcul du CRC pour justifier la valeur obtenue.

    → CRC =

12. A quel caractère correspond la valeur 0x5B ? →

    Modifier le programme de réception pour que ce caractère s'affiche →

    Faire constater

13. Modifier les programmes pour que l'information transmise soit le caractère «#».

    Vérifier que la transmission s'effectue bien et relever la valeur du CRC → CRC réel =

    Exposer en fin de fascicule comment est obtenue cette valeur.

    Faire constater

14. Donner la vitesse de transmission programmée sur le bus ? →

    Vérifier que cette vitesse est conforme sur la trame →

    Rappeler la distance maximale autorisée pour cette vitesse. →

15. Utiliser le rouleau de 80 mètres de câble Ethernet pour séparer les 2 cartes (utiliser 2 PC distants). Vérifier si le montage fonctionne toujours. Quelle précaution faut-il prendre ?

    →

    Faire constater

Document ANNEXE 1

Programme « send »

```
// demo: CAN-BUS Shield, send data
#include <mcp_can.h>
#include <SPI.h>

// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;

MCP_CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
    Serial.begin(115200);

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))                    // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}

unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};
void loop()
{
    // send data:  id = 0x00, standrad frame, data len = 8, stmp: data buf
    CAN.sendMsgBuf(0x00, 0, 8, stmp);
    delay(100);                       // send data per 100ms
}

/*********************************************************************
  END FILE
*********************************************************************/
```

Document ANNEXE 2

Programme « receive_check »

```
// demo: CAN-BUS Shield, receive data with check mode
// send data coming to fast, such as less than 10ms, you can use this way
// loovee, 2014-6-13


#include <SPI.h>
#include "mcp_can.h"


// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;

MCP_CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
    Serial.begin(115200);

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))                    // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}


void loop()
{
    unsigned char len = 0;
    unsigned char buf[8];

    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {
        CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf

        unsigned char canId = CAN.getCanId();
```

```
      Serial.println("-----------------------------");
      Serial.println("get data from ID: ");
      Serial.println(canId);

      for(int i = 0; i<len; i++)    // print the data
      {
         Serial.print(buf[i]);
         Serial.print("\t");
      }
      Serial.println();
   }
}

/*********************************************************************
  END FILE
*********************************************************************/
```