DOC CAN

Table des matières

Mise en place 2
Initialisation 2
Envoyer/Recevoir 3
    Envoyer 3
    Recevoir 3
    Opérateurs << et >> 4
Filtrage/Priorités 4
Exemples 4

Mise en place

https://os.mbed.com/teams/ADVES/code/CAN_Hello//file/09d564da0e24/main.cpp/

Il faut dupliquer le fichier contenu dans le lien et supprimer le code d'exemple.

Si on part d'un projet mbed 2 vierge il faudra tout de même copier les fichiers CANMsg et CANMsg.lib du projet précedent dans le projet. Dans les options de compilation (mbed.bld) il faut changer le lien et mettre celui-ci plus récent :

https://os.mbed.com/users/mbed_official/code/mbed/builds/3a7713b1edbc

Initialisation

On initialise un objet can de la classe CAN sur les broches adaptées du F303K8

```
CAN                 can(PA_11, PA_12);
```

On initialise ensuite des objets messages en fonction des besoins. Ici rx va être utilisé pour recevoir et tx pour envoyer des messages sur le bus. Chaque objet a plusieurs composantes comme l'id(rxmsg.id), le contenu(rxmsg.data) ou la longueur(rxmsg.len).

```
CANMsg              rxMsg;
CANMsg              txMsg;
```

On règle la fréquence de chaque nœud can. Toutes les fréquences doivent être égales pour que la communication fonctionne.

```
can.frequency(1000000);
```

On va ensuite attacher une interruption à l'objet can. A chaque fois qu'un message va être détecté sur le bus cette interruption va se déclencher (ici la fonction onCanReceived se déclenche à chaque message passé sur le bus).

```
can.attach(onCanReceived);
```

On peut utiliser la fonction filtre pour faire en sorte que l'ISR se déclenche seulement en fonction de certaines adresses de message.

```
can.filter(RX_ID, 0x7FF, CANStandard, 0);
```

```
// 1er paramètre : c'est l'id du message que l'on veut garder.
   2ème parametre : c'est le masque. Il permet de filtrer non pas un message
précis mais plusieurs messages (exemple plus bas).
   3ème parametre : le type de message typiquement CANStandard
   4ème parametre : le filtre que l'on a défini


can.filter(0b01110101101,0b11100101100,CANStandard,0);
//ce filtre par exemple ne garde que les message avec exactement 011**1*11**
comme id.
```

Le traitement des messages en fonction des id se fait donc directement dans le code par exemple dans la fonction d'interruption.

Envoyer/Recevoir

Envoyer

Pour envoyer des messages sur le bus il faut gérer son id et son contenu.

On peur le faire directement à la déclaration :

```
CANMsg              rxMsg(0x100,"Hello");
//l'id du message est 0x100 et le contenue « Hello ».
```

Ou plus tard dans le code :

```
txMsg.clear();               // on efface le contenue et l'id du msg
txMsg.id = 0x102;
txMsg<<0x5A;                  // on rentre 0x5A dans la data du msg
txMsg.len=1;                  // cette ligne n'est pas nécessaire puisque les
opérateurs << ou >> gèrent automatiquement la longueur du msg.
```

```
for(int i = 0; i < SIZE; i++){
    txmsg.data[i] = tab[i];
}//autre manière de rentrer une data.
```

Attention : la taille maximum du contenu d'un message (data) est de 8 octets. L'ID quand a lui est codé sur 11 bits pour un message CAN standard et 29 pour un CAN étendu.

Recevoir

Une fois l'ISR déclenché il faut traiter le message.

```
can.read(rxMsg); //on lit le msg et on le range dans rxmsg
```

Ensuite on trie les messages en fonction de l'id. On peut par exemple définir :

```
#define ID_MESSAGE_CAPTEUR1 0x5C
```

Puis on traite dans l'ISR le message de ce capteur si il nous intéresse.

```
if(rxmsg.id==ID_MESSAGE_CAPTEUR1){
    for(int i=0;i<rxmsg.len;i++)
        pc.printf("%x",rxmsg.data[i]);
}
```

Il faut bien sûr que le nœud du capteur envoie bien le message avec la bonne id.

Opérateurs << et >>

Ces opérateurs sont utiles pour gérer plus facilement le contenu d'un message comme vu précédemment et dans les exemples. Ils gèrent également la longueur du message.

Il faut par contre toujours inclure et exclure les données dans le même ordre pour que les données restent cohérentes.

```
            txMsg << counter;   // one byte
            txMsg << voltage;   // four bytes


            rxMsg >> counter;
            rxMsg >> voltage;
```

Filtrage/Priorités

Le protocole intègre une priorité de base pour le déplacement fluide sur le bus CAN ainsi les msg avec l'ID le plus faible seront prioritaires en cas de collision. En revanche vu la fréquence du bus ce n'est utile pour des applications normales que dans peu de cas. Ainsi la gestion des priorités des messages se fera surtout dans le code(dans l'interruption et avec la méthode .filter() ;

Exemples

Le 1er nœud envoi un msg sur le Bus CAN lorsque BP1 est enfoncé.

Le 2ème nœud va ensuite renvoyer la valeur de son potentiomètre une fois ce message reçu vers le 1er nœud. C'est un échange de message.

```
#include "mbed.h"
#include "CANMsg.h"
const unsigned int  RX_ID = 0x101;
const unsigned int  TX_ID = 0x100;
Serial pc(USBTX, USBRX);
CAN can(PA_11,PA_12);
CANMsg  rxMsg;
CANMsg  txMsg;
InterruptIn BP1(PF_1);
Timer antirebond;
float c;
void init();
void signal();
void onCanReceived();
int main(){
    init();
    while(true);
}
void init(){
    BP1.fall(&signal);
    BP1.mode(PullUp);
    pc.baud(9600);
    can.frequency(100000);
    can.attach(onCanReceived);
    can.filter(0x100,0x7FF,CANStandard,0);
    antirebond.start();
}
void signal(void){
    if(antirebond.read()>0.2){
        txMsg.clear();
        txMsg.id=TX_ID;
        if(can.write(txMsg)){
            pc.printf("Demande Envoyee\n\r");
        }
        else
            pc.printf("Demande Echouee\n\r");
    antirebond.reset();
    }
}
void onCanReceived(){
    can.read(rxMsg);
    rxMsg >> c;
    pc.printf("%f\n\r",c);
}
```

```
#include "mbed.h"
#include "CANMsg.h"

const unsigned int  RX_ID = 0x100;
const unsigned int  TX_ID = 0x101;
float a,c;
Serial pc(USBTX, USBRX);
CAN can(PA_11,PA_12);
CANMsg  rxMsg;
CANMsg  txMsg;
AnalogIn pot(D6);

void onCanReceived(void);
void init(void);
int main()
{
    init();
    while(1);
}
void onCanReceived(void)
{
    can.read(rxMsg);
    if(rxMsg.id==RX_ID){
        a=pot.read();
        txMsg.clear();
        txMsg << a;
        txMsg.id=TX_ID;
        if(can.write(txMsg)){
            pc.printf("Demande Traite\n\r");

        }
        else
            pc.printf("Erreur de Transmission\n\r");
    }
}
void init(void)
{
    pc.baud(9600);
    can.frequency(100000);
    can.filter(0x100,0x7FF,CANStandard,0);
    can.attach(onCanReceived);
}
```