TP4

Communication CAN entre des ECU basés sur le dsPIC30F4013 sous MikroC Pro sur la carte easyDsPIC7

L'objectif de ce TP est de mettre en œuvre une communication CAN entre des ECU basés sur le dsPIC30F4013. Ce projet illustre le fonctionnement d'un bus CAN dans un système embarqué distribué. La figure 1 ci-jointe illustre le premier montage à mettre en œuvre.

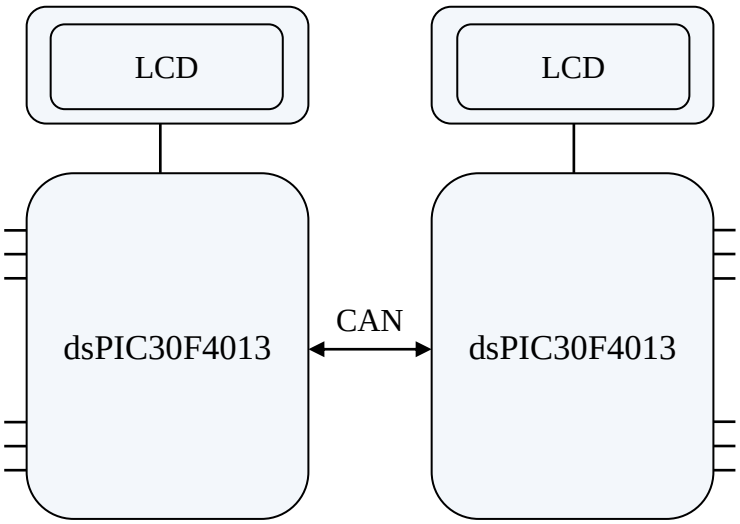

Figure 1. Communication entre deux cartes dsPIC30F4013 via le protocole CAN

Dans un premier temps, la communication entre deux ECU sera mise en œuvre. L'environnement de développement utilisé est MikroC for dsPIC. Pour cela, deux programmes distincts ont été élaborés : l'un destiné à la première ECU, et l'autre à la seconde ECU.

Le premier programme est nommé **Can_1st()**, tandis que le second c'est **Can_2nd()**. Ces deux programmes permettent d'échanger des données en temps réel via CAN. Les deux programmes exploitent une bibliothèque CAN développé dans l'environnement de travail de MikroC.

Programme du Can_1st

```
unsigned int Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags;          // can flags
unsigned int Rx_Data_Len;                                            // received data length in bytes
char RxTx_Data[8];                                                   // can rx/tx data buffer
char Msg_Rcvd;                                                       // reception flag
const unsigned long ID_1st = 12111, ID_2nd = 3;                      // node IDs
unsigned long Rx_ID;
```

```
void main() {

  ADPCFG = 0xFFFF;
  PORTB = 0;
  TRISB = 0;

  Can_Init_Flags = 0;                    //
  Can_Send_Flags = 0;                    // clear flags
  Can_Rcv_Flags  = 0;                    //

  Can_Send_Flags = _CAN_TX_PRIORITY_0 &                  // form value to be used
                   _CAN_TX_XTD_FRAME &                   // with CAN1Write
                   _CAN_TX_NO_RTR_FRAME;

  Can_Init_Flags = _CAN_CONFIG_SAMPLE_THRICE &           // form value to be used
                   _CAN_CONFIG_PHSEG2_PRG_ON &           // with CAN1Initialize
                   _CAN_CONFIG_XTD_MSG &
                   _CAN_CONFIG_DBL_BUFFER_ON &
                   _CAN_CONFIG_MATCH_MSG_TYPE &
                   _CAN_CONFIG_LINE_FILTER_OFF;

  RxTx_Data[0] = 9;                                      // set initial data to be sent
  CAN1Initialize(1,3,3,3,1,Can_Init_Flags);              // initialize CAN1

  CAN1SetOperationMode(_CAN_MODE_CONFIG,0xFF);      // set CONFIGURATION mode

  CAN1SetMask(_CAN_MASK_B1,-1,_CAN_CONFIG_MATCH_MSG_TYPE &
_CAN_CONFIG_XTD_MSG);                                    // set all mask1 bits to ones
  CAN1SetMask(_CAN_MASK_B2,-1,_CAN_CONFIG_MATCH_MSG_TYPE &
_CAN_CONFIG_XTD_MSG);                                    // set all mask2 bits to ones
  CAN1SetFilter(_CAN_FILTER_B2_F3,ID_2nd,_CAN_CONFIG_XTD_MSG);
// set id of filter B1_F1 to 2nd node ID

  CAN1SetOperationMode(_CAN_MODE_NORMAL,0xFF);           // set NORMAL mode

  CAN1Write(ID_1st, RxTx_Data, 1, Can_Send_Flags);       //

  while(1) {                                             //
    Msg_Rcvd = CAN1Read(&Rx_ID , RxTx_Data , &Rx_Data_Len, &Can_Rcv_Flags);
                                                         // receive message
    if ((Rx_ID == ID_2nd) && Msg_Rcvd) {                 //
      PORTB = RxTx_Data[0];                              //
      RxTx_Data[0]++;                                    //
      Delay_ms(10);
      CAN1Write(ID_1st, RxTx_Data, 1, Can_Send_Flags);   // incremented data back
    }
  }
}
```

Programme du can_2nd

```
unsigned int Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags;                  // can flags
unsigned int Rx_Data_Len;                                                    // received data length in bytes
char RxTx_Data[8];                                                           // can rx/tx data buffer
char Msg_Rcvd;                                                               // reception flag
const unsigned long ID_1st = 12111, ID_2nd = 3;                              // node IDs
unsigned long Rx_ID;

void main() {

  ADPCFG = 0xFFFF;
  PORTB = 0;
  TRISB = 0;

  Can_Init_Flags = 0;                                                        //
  Can_Send_Flags = 0;                                                        // clear flags
  Can_Rcv_Flags  = 0;                                                        //

  Can_Send_Flags = _CAN_TX_PRIORITY_0 &                                      // form value to be used
                   _CAN_TX_XTD_FRAME &                                       // with CAN1Write
                   _CAN_TX_NO_RTR_FRAME;

  Can_Init_Flags = _CAN_CONFIG_SAMPLE_THRICE &                               // form value to be used
                   _CAN_CONFIG_PHSEG2_PRG_ON &                               // with CAN1Initialize
                   _CAN_CONFIG_XTD_MSG &
                   _CAN_CONFIG_DBL_BUFFER_ON &
                   _CAN_CONFIG_MATCH_MSG_TYPE &
                   _CAN_CONFIG_LINE_FILTER_OFF;

  CAN1Initialize(1,3,3,3,1,Can_Init_Flags);                                  // initialize CAN

  CAN1SetOperationMode(_CAN_MODE_CONFIG,0xFF);                               // set CONFIGURATION mode

  CAN1SetMask(_CAN_MASK_B1,-1,_CAN_CONFIG_MATCH_MSG_TYPE &
_CAN_CONFIG_XTD_MSG);                                                        // set all mask1 bits to ones
  CAN1SetMask(_CAN_MASK_B2,-1,_CAN_CONFIG_MATCH_MSG_TYPE &
_CAN_CONFIG_XTD_MSG);                                                        // set all mask2 bits to ones
  CAN1SetFilter(_CAN_FILTER_B1_F1,ID_1st,_CAN_CONFIG_XTD_MSG);
                                                                             // set id of filter B1_F1 to 1st node ID

  CAN1SetOperationMode(_CAN_MODE_NORMAL,0xFF);                               // set NORMAL mode

  while(1) {                                                                 // endless loop
    Msg_Rcvd = CAN1Read(&Rx_ID , RxTx_Data , &Rx_Data_Len, &Can_Rcv_Flags);
                                                                             // receive message
    if ((Rx_ID == ID_1st) && Msg_Rcvd) {                                     // if message received check id
```

```
    PORTB = RxTx_Data[0];                          // id correct, output data at PORTB
    RxTx_Data[0]++;                                // increment received data
    CAN1Write(ID_2nd, RxTx_Data, 1, Can_Send_Flags);   // send incremented data back
   }
  }
}
```

1. Exécutez les deux programmes Can_1st() et Can_2nd() dans MikroC Pro for dsPIC, puis charger les fichiers binaires sur deux cartes de développement easydsPIC30F4013.
   Interprétez le fonctionnement global de ces deux codes.
   Déterminer les identifiants (IDs) utilisés dans le programme pour chacune des deux ECUs.

2. Modifiez les deux programmes Can_1st() et Can_2nd() afin d'afficher, sur les écrans LCD intégrés de chaque carte dsPIC30F4013, les valeurs échangées via le bus CAN.
   Expliquez ensuite les résultats observés sur chaque écran LCD.
   Utilisez pour cela le configuration suivante du module LCD :

```
// LCD module connections
sbit LCD_RS at LATD0_bit;
sbit LCD_EN at LATD1_bit;
sbit LCD_D4 at LATB0_bit;
sbit LCD_D5 at LATB1_bit;
sbit LCD_D6 at LATB2_bit;
sbit LCD_D7 at LATB3_bit;

sbit LCD_RS_Direction at TRISD0_bit;
sbit LCD_EN_Direction at TRISD1_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End LCD module connections
```

3. Modifiez les deux programmes Can_1st() et Can_2nd() pour qu'ils lisent une valeur analogique sur le canal AN10 à l'aide de l'instruction ADC_Read(10), puis afficher sur l'écran LCD la valeur locale mesurée (par le convertisseur analogique/numérique) sur la première ligne, et la valeur reçue via le bus CAN sur la deuxième ligne.

4. On désir réaliser un réseaux CAN composé de 4 ECUs.
   La première représente un BSI qui affiche les valeurs des capteurs, la deuxième est une ECU qui fournit la valeur du capteur infrarouge, la troisième ECU fournit la vitesse véhicule et la quatrième ECU est un calculateur ACC.

- Affecter des IDs aux quatre calculateurs afin de garantir une priorité convenable.

- Proposer un modèle de communication entre les ECUs (qui envoie et quand : par appel, de manière périodique,...).

- Ecrire le programme des quatre ECUs et charger les fichiers binaires sur les cartes easydsPIC7.