TP6 M2 ESE Electronique M2 USTO 2020/21

TP6 M2 ESE BUS CAN en MicoC PRO ***avec affichage USTO 2020***

Le protocole CAN (Control Area Network) est un protocole de communication série qui supporte des systèmes temps réel avec un haut niveau de fiabilité.
Structure d'un réseau CAN: Pour envoyer ou recevoir
Le microcontrôleur est connecté au bus CAN à l'aide d'un contrôleur externe MCP2515 via SPI (Serial Peripheral Interface) et d'un transceiver MCP2551 pour adaptation au bus can, constitué d'une paire de fils torsadés terminés par une résistance de 120 ohm à chacun de ses bouts. Le maître SPI communique avec un esclave à l'aide de l'horloge série (SCK), master out slave in (MOSI), master in slave out (MISO) and slave select (SS)

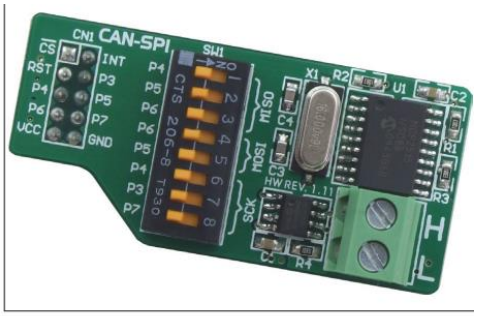
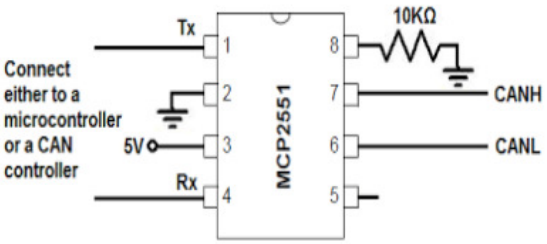
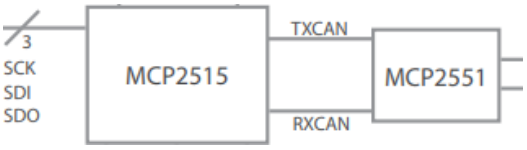


Figure 1: CAN-SPI additional board





En bus CAN, le masque détermine les bits qui seront vérifiés. Le filtre sert à accepter la trame si le filtre est identique à la trame entrante. Donc, si on veux filtrer sur tes 11 bits(l'identifiant complet), le masque doit être configuré à "1" pour les 11 bits et le filtre doit être configuré à "X" pour ne laisser entrer que les trames avec l'identifiant X

.

//Code pour 1er nœud CAN :

```
unsigned char Can_Init_Flags, Can_Send_Flags,
Can_Rcv_Flags; // can flags
unsigned char Rx_Data_Len;  // taille data reçues
char RxTx_Data[8];      // can rx/tx data buffer
char Msg_Rcvd;          //flag de reception
const long ID_1st = 12111, ID_2nd = 3;
// IDs des 2 noeuds
long Rx_ID;
// connexions du module CANSPI et pic
sbit CanSpi_CS          at RC0_bit;
sbit CanSpi_CS_Direction at TRISC0_bit;
sbit CanSpi_Rst         at RC2_bit;
sbit CanSpi_Rst_Direction at TRISC2_bit;

void main() {
   PORTB = 0;
   TRISB = 0;
  Can_Init_Flags = 0;
  Can_Send_Flags = 0;       // clear flags
  Can_Rcv_Flags  = 0;
  Can_Send_Flags = _CANSPI_TX_PRIORITY_0 &
_CANSPI_TX_XTD_FRAME &
_CANSPI_TX_NO_RTR_FRAME;
   // valeur a utiliser avec CANSPIWrite
```

```
Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE
&     _CANSPI_CONFIG_PHSEG2_PRG_ON &
_CANSPI_CONFIG_XTD_MSG &
_CANSPI_CONFIG_DBL_BUFFER_ON &
_CANSPI_CONFIG_VALID_XTD_MSG;
  // Form value to be used// with CANSPIInit
  SPI1_Init();     // initialiser module SPI1
  CANSPIInitialize(1,3,3,3,1,Can_Init_Flags);
 // Initialize external CANSPI module
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF);
// set CONFIGURATION mode
  CANSPISetMask(_CANSPI_MASK_B1,-1,
_CANSPI_CONFIG_XTD_MSG);
// mettre tous les bits du mask1 a 1
  CANSPISetMask(_CANSPI_MASK_B2,-1,
_CANSPI_CONFIG_XTD_MSG);
// mettre tous les bits du mask2 a 1
CANSPISetFilter(_CANSPI_FILTER_B2_F4,ID_2nd,
_CANSPI_CONFIG_XTD_MSG);
// set id of filter B2_F4 to 2nd node ID
CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF);
// set mode NORMAL
  RxTx_Data[0] = 9;       //donnée initiale a envoyer
  CANSPIWrite(ID_1st, RxTx_Data, 1, Can_Send_Flags);
// envoie du message initial
  while(1) {
Msg_Rcvd = CANSPIRead(&Rx_ID, RxTx_Data ,
&Rx_Data_Len, &Can_Rcv_Flags);// receive message
    if ((Rx_ID == ID_2nd) && Msg_Rcvd) {
// si message reçu verifier ID
      PORTB = RxTx_Data[0];
//si id correct, afficher data sur PORTB
      RxTx_Data[0]++ ;    // incrementer données reçues
Delay_ms(10);
      CANSPIWrite(ID_1st, RxTx_Data, 1, Can_Send_Flags);
// envoie des données incrementées
    } }}
```

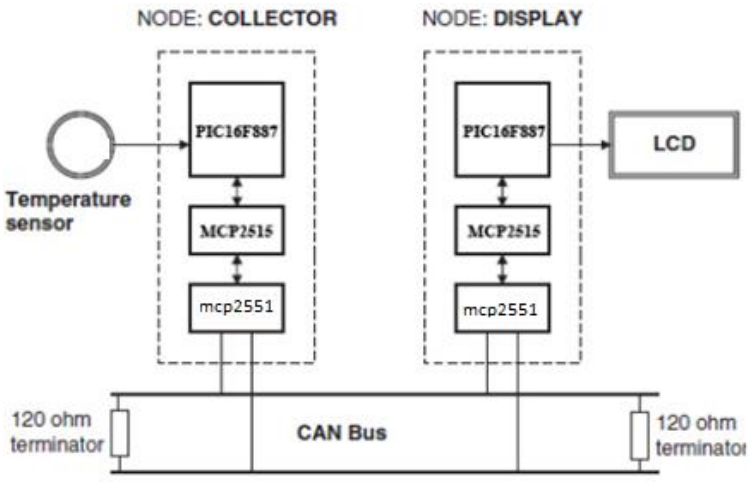
```
//Code pour deuxième CAN node:
unsigned char Can_Init_Flags, Can_Send_Flags,
Can_Rcv_Flags; // can flags
unsigned char Rx_Data_Len;
 // taille des données reçues en octet
 char RxTx_Data[8];
  //buffer des données can rx/tx
char Msg_Rcvd;     // reception flag
const long ID_1st = 12111, ID_2nd = 3;
//identifications des noeuds: node IDs
long Rx_ID;
// connexion du module CANSPI
sbit CanSpi_CS          at      RC0_bit;
sbit CanSpi_CS_Direction    at  TRISC0_bit;
sbit CanSpi_Rst         at      RC2_bit;
sbit CanSpi_Rst_Direction   at  TRISC2_bit;

void main() {
  PORTB = 0;
  TRISB = 0;
  Can_Init_Flags = 0;
  Can_Send_Flags = 0;  // clear flags
  Can_Rcv_Flags  = 0;                                  //
  Can_Send_Flags = _CANSPI_TX_PRIORITY_0 &
                   _CANSPI_TX_XTD_FRAME &
                   _CANSPI_TX_NO_RTR_FRAME;
// former la valeur a utiliser avec CANSPIWrite
  Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE &
_CANSPI_CONFIG_PHSEG2_PRG_ON &
_CANSPI_CONFIG_XTD_MSG &
_CANSPI_CONFIG_DBL_BUFFER_ON &
  _CANSPI_CONFIG_VALID_XTD_MSG &
_CANSPI_CONFIG_LINE_FILTER_OFF;
   // Form value to be used// with CANSPIInit
SPI1_Init();       // initialiser module SPI1
  CANSPIInitialize(1,3,3,3,1,Can_Init_Flags);
// initialize external CANSPI module
CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF);
// set CONFIGURATION mode
  CANSPISetMask(_CANSPI_MASK_B1,-1,
_CANSPI_CONFIG_XTD_MSG);
 // mettre tous les bits du mask1 a 1
  CANSPISetMask(_CANSPI_MASK_B2,-1,
_CANSPI_CONFIG_XTD_MSG);
// mettre tous les bits du mask2 a 1
CANSPISetFilter(_CANSPI_FILTER_B2_F3,ID_1st,_CA
NSPI_CONFIG_XTD_MSG);
// mettre le filtre d'id B2_F3 à ID du 1ier noeud
CANSPISetOperationMode(_CANSPI_MODE_NORMAL,
0xFF);            // set NORMAL mode
  while (1) {
Msg_Rcvd = CANSPIRead(&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 PORTC
  RxTx_Data[0]++ ;      // increment received data
 CANSPIWrite(ID_2nd, RxTx_Data, 1, Can_Send_Flags);
  // send incremented data back
    } }}
```

Le mikroC PRO for PIC fournit une bibliothèque (pilote) pour travailler avec le module CAN.

Le Programme : une simple démonstration de l'utilisation des routines de la bibliothèque CAN. Le premier nœud initie la communication avec le deuxième nœud en envoyant des données à son adresse. Le second noeud répond en renvoyant les données incrémentées de 1. Le premier noeud procède ensuite de la même manière et renvoie les données incrémentées au second nœud, etc.

Réaliser la connexion entre deux nœuds a base du PIC16F877A sur la carte easypic5 via le bus can en utilisant deux modules CANSPI sur les ports C des deux cartes easypic Le premier nœud implémente le premier programme et le deuxième nœud le IIème programme



Utiliser un LCD pour visualiser les données inter changées. Ajouter le programme adéquat.

Compte rendu

Etudier en expliquant les différentes fonctions CANSPI du microcPro

Réaliser l'application donnée sur le schéma ci contre en prenant le capteur de température DS 1820 . Donner le schéma détaillé avec les programmes

Faites la conception complète du nœud émetteur et du nœud récepteur