Systèmes Embarqués II

Communication CAN

ISAT – EPHEC 2020-2021

Juan Alvarez et Olivier Grabenweger

2 Table des matières

...

1 Introduction

Dans ce TP, nous utilisons la communication CAN afin d'envoyer ou recevoir une trame.
Le matériel utilisé est le suivant (pour 1 utilisateur) :

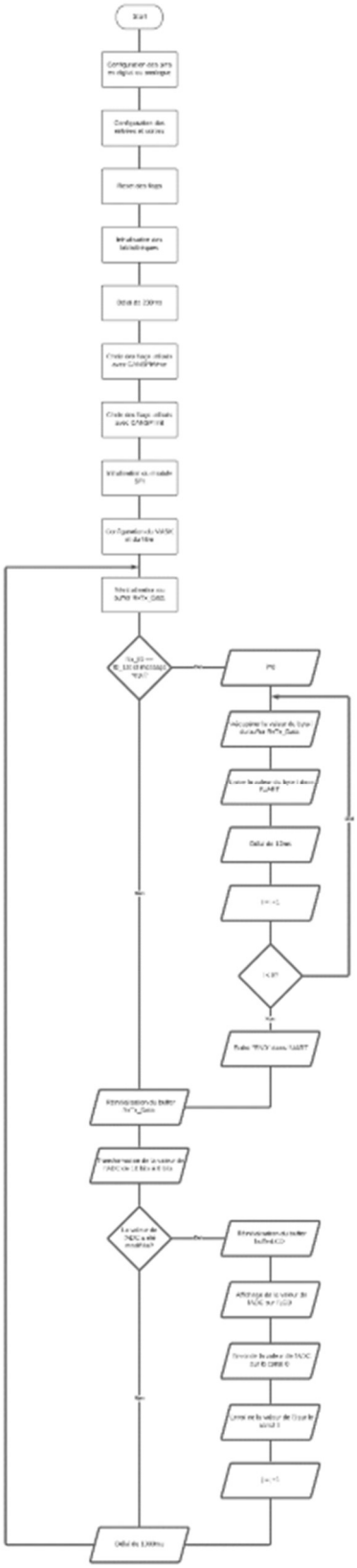
- 1 PIC18F45K22
- 1 Ecran LCD
- 2 Boutons Poussoir
- 1 ADC
- 1 Câble CAN
- 1 UART

(En sachant qu'on a besoin de minimum 2 utilisateurs donc le matériel doit être doublé).

Nous avons 2 compteurs qui s'incrémentent/décrémentent.
Le premier compteur voit sa valeur changer à l'aide d'un ADC et est affiché en temps réel sur un LCD. Le second compteur est contrôlé à l'aide de 2 boutons poussoirs. Nous utilisons le protocole de communication CAN afin d'envoyer nos valeurs sous formes de trames vers un autre utilisateur.
Sur notre moniteur série, nous affichons les données, préalablement traitées, que nous recevons de ce même utilisateur.

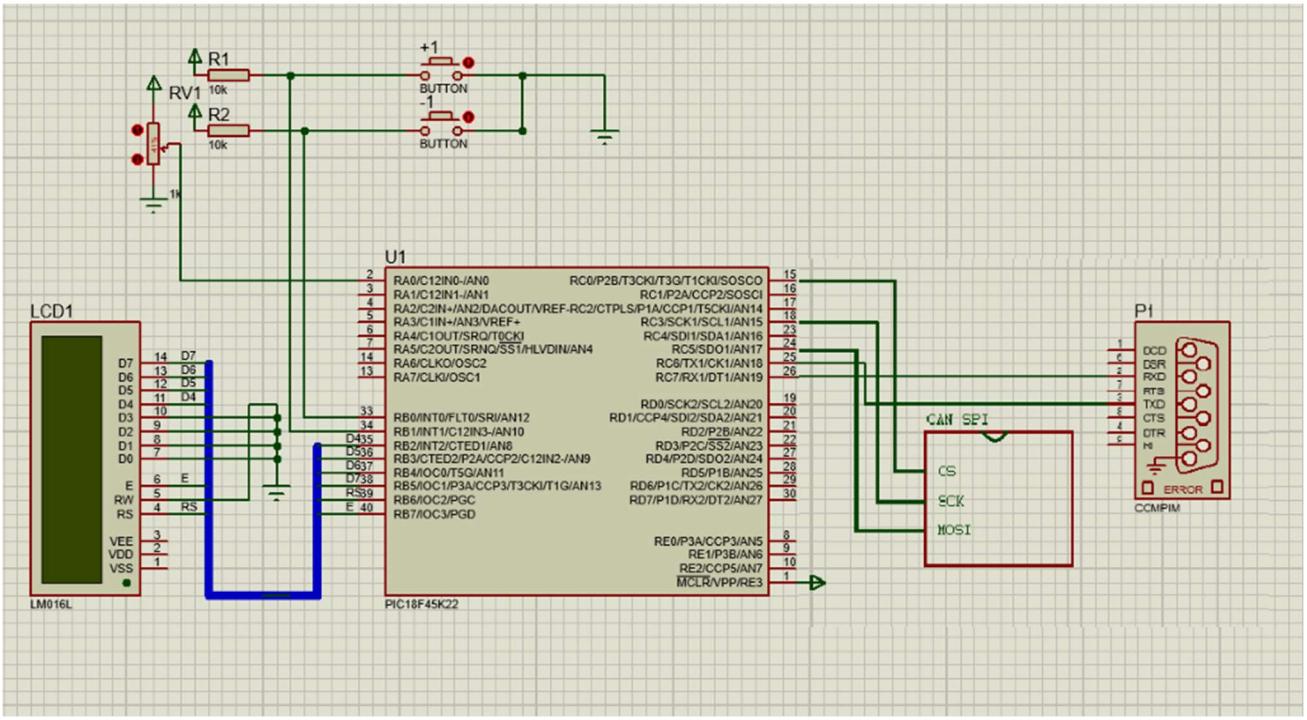
...

2 Organigramme – Mindmapping



...

3 Schéma de câblage



4 Code source

```
//
// Created by Juan & Oli on 5/14/2020.
//

unsigned char Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags; // can flags
unsigned char Rx_Data_Len;                                   // received data length in
bytes
char RxTx_Data[8] = {0};                                     // can rx/tx data buffer
char Msg_Rcvd;                                               // reception flag
const long ID_1st = 12111, ID_2nd = 3;                       // node IDs
long Rx_ID;
int recu;
int i,j,cpt,oldcpt;
char bufferLCD[16];
int nombre,oldnbr;
char bufferUART[16];

// LCD module connections
sbit LCD_RS at RB6_bit;
sbit LCD_EN at RB7_bit;
sbit LCD_D4 at RB2_bit;
sbit LCD_D5 at RB3_bit;
sbit LCD_D6 at RB4_bit;
```

...

```
24  sbit LCD_D7 at RB5_bit;
25  sbit LCD_RS_Direction at TRISB4_bit;
26  sbit LCD_EN_Direction at TRISB5_bit;
27  sbit LCD_D4_Direction at TRISB0_bit;
28  sbit LCD_D5_Direction at TRISB1_bit;
29  sbit LCD_D6_Direction at TRISB2_bit;
30  sbit LCD_D7_Direction at TRISB3_bit;
31  // End LCD module connections
32
33  // CANSPI module connections
34  sbit CanSpi_CS           at  RC0_bit;
35  sbit CanSpi_CS_Direction  at  TRISC0_bit;
36  sbit CanSpi_Rst          at  RC2_bit;
37  sbit CanSpi_Rst_Direction at  TRISC2_bit;
38  // End CANSPI module connections
39
40  void Interrupt()
41  {
42     //External Interrupt
43     if(INT0F_bit)
44     {
45        INT0F_bit = 0;
46        cpt++;
47     }
48     if(INT1F_bit)
49     {
50        INT1F_bit = 0;
51        cpt--;
52     }
53  }
54
55  void main()
56  {
57    ANSELA = 0x01;                  // Configure AN pins as digital I/O
58    ANSELB = 0;
59    ANSELC = 0;
60    ANSELD = 0;
61
62    C1ON_bit = 0;              // Disable comparators
63    C2ON_bit = 0;
64
65    TRISD = 0x00;
66    TRISA = 0x01;                   // OUTPUT = 0 Iput = 1
67    TRISB = 0x03;
68    TRISC = 0x00;
69    Can_Init_Flags = 0;                                //
70    Can_Send_Flags = 0;                                  // clear flags
```

...

```
71   Can_Rcv_Flags  = 0;                                //
72   ADC_Init();
73   Lcd_Init();
74
75  UART1_Init(9600);
76
77   Delay_ms(200);
78
79   Can_Send_Flags = _CANSPI_TX_PRIORITY_0 &                // form value to be
80  used
81                 _CANSPI_TX_XTD_FRAME &                  //     with CANSPIWrite
82                 _CANSPI_TX_NO_RTR_FRAME;
83
84   Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE &          // Form value
85  to be used
86                 _CANSPI_CONFIG_PHSEG2_PRG_ON &          //  with CANSPIInit
87                 _CANSPI_CONFIG_XTD_MSG &
88                 _CANSPI_CONFIG_DBL_BUFFER_ON &
89                 _CANSPI_CONFIG_VALID_XTD_MSG;
90
91   SPI1_Init();                                    // initialize SPI1 module
92
93   CANSPIInitialize(1,3,3,3,1,Can_Init_Flags);              // Initialize external
94  CANSPI module
95   CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF);          // set
96  CONFIGURATION mode
97   CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_XTD_MSG);       //
98  set all mask1 bits to ones
99   CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_XTD_MSG);       //
100 set all mask2 bits to ones
101  CANSPISetFilter(_CANSPI_FILTER_B2_F4,ID_1st,_CANSPI_CONFIG_XTD_MSG);//
102 set id of filter B2_F4 to 2nd node ID
103
104  CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF);          // set
105 NORMAL mode
106
107  //RxTx_Data[0] = {};                                 // set initial data to be sent
108   Lcd_Cmd(_LCD_CLEAR);
109   Lcd_Out(1,1,"yol0");
110
111  //External
112    INTEDG0_bit = 1;
113    INT0E_bit = 0;
114    INT0E_bit = 1;
115    PEIE_bit  = 1;              // Enable peripheral interrupts
116    GIE_bit   = 1;              // Enable GLOBAL interrupts
117
```

...

```
118  for(;;)
119    {
120    memset(Rxtx_Data,0,sizeof(RxTx_Data));
121      Msg_Rcvd = CANSPIRead(&Rx_ID , RxTx_Data , &Rx_Data_Len,
122 &Can_Rcv_Flags);// receive message
123      if ((Rx_ID == ID_1st) && Msg_Rcvd) // if message received check id
124        {
125
126        for(i=0; i < 8 ; i++)
127        {
128          recu = RxTx_Data[i];
129          //memset(bufferUART,0,sizeof(bufferUART));
130
131          sprintf(bufferUART,"nbr rcvd : %04d",recu);
132          UART1_Write_Text(bufferUART);
133          UART1_Write_Text("\n\r");
134          delay_ms(10);
135
136        }
137        UART1_Write_Text("end");
138        UART1_Write_Text("\n\r");
139      }
140      //}
141      memset(Rxtx_Data,0,sizeof(RxTx_Data));
142      nombre = ADC_Read(0) >> 2;
143
144      if((oldnbr != nombre) | | (oldcpt != cpt))
145      {
146      // vérifie nouvelle valeur pot
147        oldnbr = nombre;
148        oldcpt = cpt;
149        memset(bufferLCD,0,sizeof(bufferLCD));
150        sprintf(bufferLCD,"nbr:%04d & cpt : %04d", nombre,cpt);
151        Lcd_Out(2,1,bufferLCD);
152        Lcd_Cmd(_LCD_CURSOR_OFF);
153        RxTx_Data [0] = nombre;
154        RxTx_Data[1] = cpt;
155        CANSPIWrite(ID_2nd, RxTx_Data, 8, Can_Send_Flags);     // send data
156        j++;
157      }
158      delay_ms(1000);
159    }
160 }
```

…

Analyse du code :
Le fonctionnement du code est le suivant :

Première étape → déclaration et initialisation des différentes variables

Ensuite on règle les différents flags du CANSPI

- ➔ Une fonction très importante est la suivante :
  CANSPISetFilter (_CANSPI_FILTER_B2_F4,ID_1st,_CANSPI_CONFIG_XTD_MSG);

Les arguments sont les suivants : (Type de filtre – id à filtrer – format du msg)

Ensuite on rentre dans la boucle où :
1) Deux interruptions externes sont utilisées pour incrémenter deux compteurs
2) Lecture ADC
3) On vérifie si une valeur a changé
4)si oui on formate et on affiche la nouvelle valeur sur le LCD

Et finalement on envoie via CANSPI les deux compteurs s'ils ont changé sur le premier et le deuxième byte.

5 Conclusion

La réalisation de ce premier tp aura permis une approche rapide mais efficace du protocole CAN.
Et même si l'application de celui-ci dans ce premier tp reste assez simple, nous pensons que le travail réalisé peut être étendu à une application plus complexe.
Nous regrettons de ne pas avoir assez bien compris toutes les différentes constantes disponibles dans la configuration du protocole, mais espérons pouvoir revenir dessus ultérieurement.