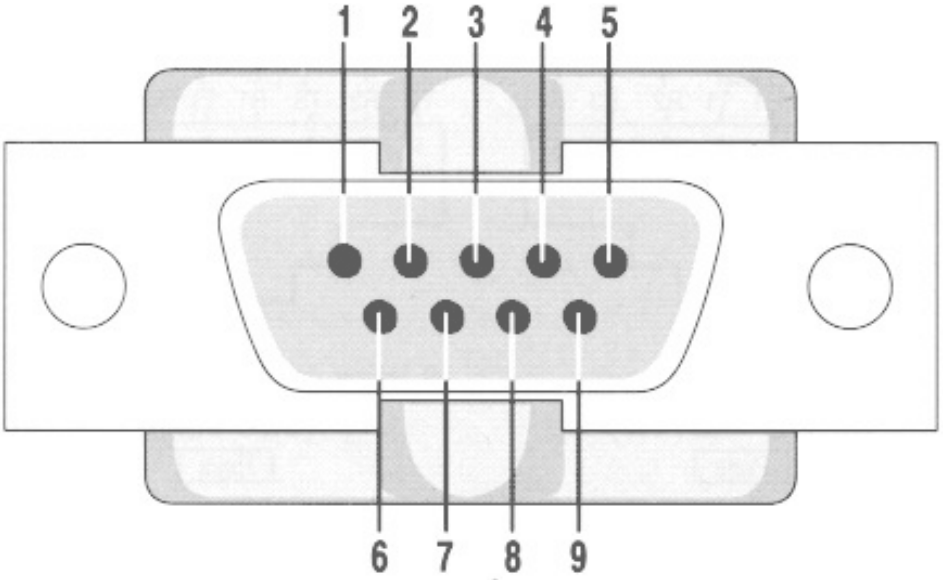
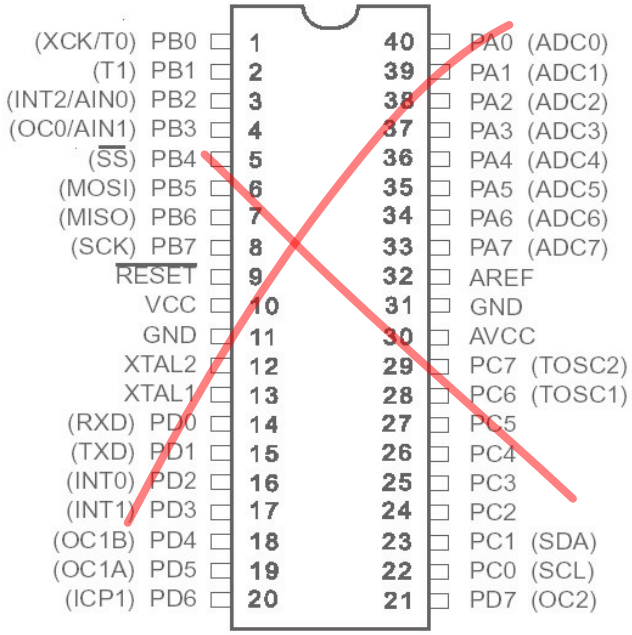
Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE




# Commande d'un ATméga par liaison RS232

*Projet d'Études et Réalisations – Semestre 4*

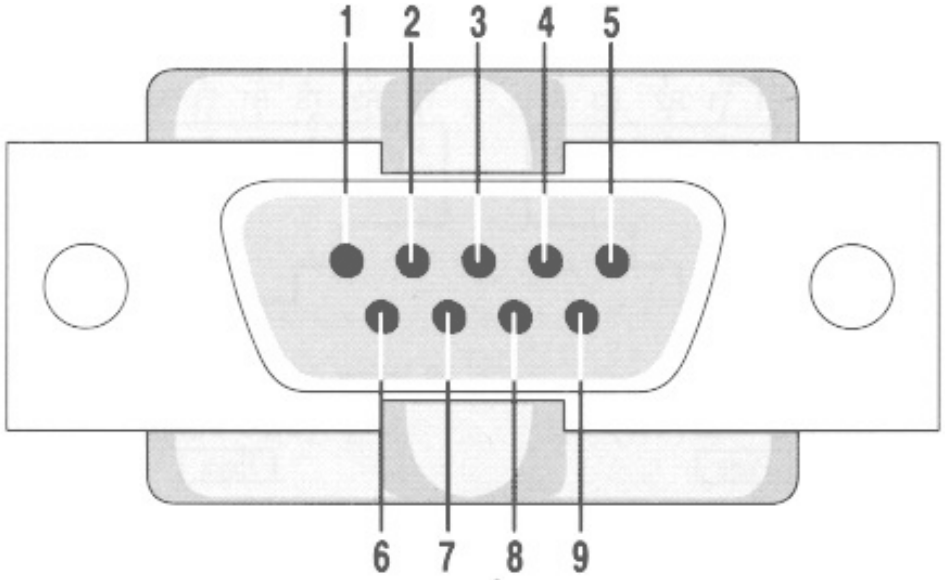



Autre chose pour illustrer projet ? Éviter le schéma...

MOREAU Thomas
AUGER Loïc
2011 – 2013
Groupe P1

Enseignants :
Thierry LEQUEU
Sofi RODIER

Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle




# Commande d'un ATméga par liaison RS232

*Projet d'Études et Réalisations – Semestre 4*

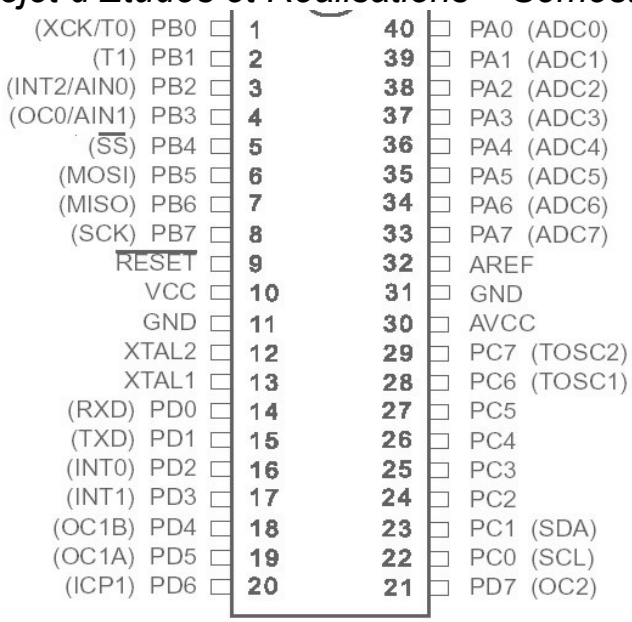



*Illustration 1: Connecteur DB9 et ATméga8535 (http://www.arcelect.com/rs232.htm)*

MOREAU Thomas
AUGER Loïc
2011 – 2013
Groupe P1

Enseignants :
Thierry LEQUEU
Sofi RODIER

# Sommaire



Fautes

Augmenter espace avant des titres
Réduire retrait [5

:

Attention à mise en page MeP qui doit structurer les infos [7 8
MeP à revoir [5 7....
Utiliser index des illustrations pour y faire référence [12
Utiliser un style caractère autre couleur autre police pour le code (ici en vert !)[13 14 15

Trop de : tue les : !!!! [7.

Attention à la gestion des listes à puces [7 8

Introduire les chapitres et sous chapitres [ 4 5 11 13

Utiliser notes de bas de page + souvent

Résumé un peu trop résumé !

Signaler dès intro par ndbdp présence index lexical, à compléter

Mettre une page de titre ANNEXES
Avec le sommaire des annexes
Sur chacune mettre index et titre (en style de titreAnnexe)
Ajouter sur chaque page des annexes, quand elles en ont plusieurs, un tête avec un champ titreAnnexe

Ajouter espace avant à TM1

# Introduction

Contexte ?
Iut/étudiant 1et 2/étude et réa/DUT...

La liaison série RS232 est utilisée dans de nombreux domaines informatiques. Elle permet la communication avec des périphériques tels qu'un modem ou un scanner. Mais elle est aussi utilisée pour programmer un automate par un PC.

Nous allons l'utiliser ici afin de communiquer avec un microcontrôleur ATméga8535. Une supervision de cette communication sera effectuée avec C++ Builder.

Dans le cadre du cours d'Études et Réalisations du 4e semestre, nous avons sélectionné comme projet la communication avec un ATméga par liaison série RS232.

Pour exposer au mieux l'élaboration de notre projet, nous ferons tout d'abord sa présentation. Dans un deuxième temps nous développerons l'étude théorique nécessaire à sa réalisation. Puis nous terminerons par sa réalisation.

Problématique ? Que cela soit opérationnel par exemple ?

# 1. Présentation

## 1.1. La liaison RS232

La liaison RS232 est un standard de transmission de données séries entres équipements. Elle à été développé dans les années 60 par l'EIA[1]. Elle était définie pour la transmission de données de type texte ASCII[2] entre les systèmes numériques et les modems.

Compte tenu de sa simplicité de mise en œuvre et des atouts de la communication numérique, l'utilisation de la liaison série fut rapidement généralisée. On la retrouve dans :

- la communication entre ordinateurs,
- la transmission de données entre ordinateur et périphériques (imprimantes, souris, claviers, modems, …),
- la communication avec tous les systèmes à microprocesseur ou microcontrôleur.

Le standard RS232 prévoit des formats de transmission synchrone et asynchrone. Dans la majorité des cas, la transmission est de type asynchrone, c'est à dire qu'elle ne transmet pas de signal horloge contrairement aux liaisons synchrones. Ce standard définit :

- le format des données transmises,
- le brochage des connecteurs (ici : DB9),
- les niveaux de tension du support physique de transmission,
- le protocole d'échange des informations.

### 1.1.1. Les modes de transmission RS232

- Simplex : mode de transmission unidirectionnel.
- Semi-duplex (ou half-duplex) : mode de transmission bidirectionnel, mais un seul dispositif peut émettre à la fois.
- Full-duplex : mode de transmission bidirectionnel. Les deux dispositifs peuvent émettre en même temps.

### 1.1.2. Description de la liaison

1 EIA : Electronic Industries Association
2 ASCII : American Standard Code for Information Interchange

Image pourrait être agrandie pour mettre touts les n° et en légende les noms et ainsi ne plus avoir besoin du tableau

ET BCP +facile à comprendre

En annexe avec renvoi vers celle-ci

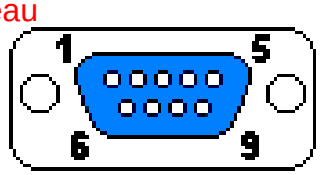



*Illustration 2: Broches du connecteur DB9 (port COM1)*

| Nom | C++ Builder | Broche | Sens |
|---|---|---|---|
| DCD | RLSD | Broche n°1 | Entrée |
| RxD | Rx | Broche n°2 | Entrée |
| TxD | Tx | Broche n°3 | Sortie |
| DTR | DTR | Broche n°4 | Entrée |
| Masse | -- | Broche n°5 | -- |
| DSR | DSR | Broche n°6 | Entrée |
| RTS | RTS | Broche n°7 | Sortie |
| CTS | CTS | Broche n°8 | Entrée |
| RI | RING | Broche n°9 | Entrée |

➢ *Tableau 1: Entrées/sorties du connecteur DB9*

Description des signaux:

Utiliser

➢ **Broche 1** : DCD ou RLSD (Data Carrier Detect) ou (Receive Line Signal Detect) cette ligne est une entrée active à l'état haut. Elle signale à l'ordinateur qu'une liaison a été établie avec un correspondant ;

➢ **Broche 2** : RxD (Receive Data) cette ligne est une entrée. C'est ici que transitent les informations du correspondant vers l'ordinateur ;

➢ **Broche 3** : TxD (Transmit Data) cette ligne est une sortie. Les données de l'ordinateur vers le correspondant sont véhiculées par son intermédiaire ;

➢ **Broche 4** : DTR (Data Terminal Ready) cette ligne est une sortie active à l'état haut. Elle permet à l'ordinateur de signaler au correspondant que le port série a été libéré et qu'il peut être utilisé s'il le souhaite ;

➢ **Broche 5** : GND (Ground) c'est la masse ;

➢ **Broche 6** : DSR (Data Set Ready) cette ligne est une entrée active à l'état haut. Elle permet au correspondant de signaler qu'une donnée est prête ;

➢ **Broche 7** : RTS (Request To Send) cette ligne est une sortie, qui quand elle est active est à l'état haut. Elle indique au correspondant que l'ordinateur veut lui transmettre des données ;

➢ **Broche 8** : CTS (Clear To Send) cette ligne est une entrée active à l'état haut. Elle indique à l'ordinateur que le correspondant est prêt à recevoir des données ;

➢ **Broche 9** : RI ou RING (Ring Indicator) cette ligne est une entrée active à l'état haut. Elle permet à l'ordinateur de savoir qu'un correspondant veut initier une communication avec lui.

Le plus souvent l'échange d'octets s'effectue sur les seuls fils RxD et TxD. Dans ce cas les signaux d'échange doivent être simulés, DSR relié à DTR et RTS relié à CTS

## 1.2. Cahier des charges

**Explication du projet** : communiquer avec un microcontrôleur ATméga8535 par PC via une liaison RS232.

**But du projet :** asservir une carte composée d'un ATméga8535 via une liaison RS232, par l'intermédiaire d'une interface graphique sur un ordinateur, développée sous C++ Builder.

**Finalité du produit :** la gestion complète de la carte, gestion des périphériques (bouton poussoir, LED, potentiomètre, écran LCD).

**Mise en œuvre** : nous réaliserons deux codes :

➢ le premier sur l'ATméga sera développé sur CodeVision AVR et permettra de recevoir la trame RS232, de l'appliquer, et d'émettre une réponse.

➢ Le second sera l'interface homme machine développé sous C++ Builder et permettra de générer une trame RS232 et devra pouvoir gérer la carte (bouton poussoir, LED, potentiomètre, écran LCD).

**Éléments du projet :**

➢ Microcontrôleur ATméga8535

➢ Programmateur d'ATméga

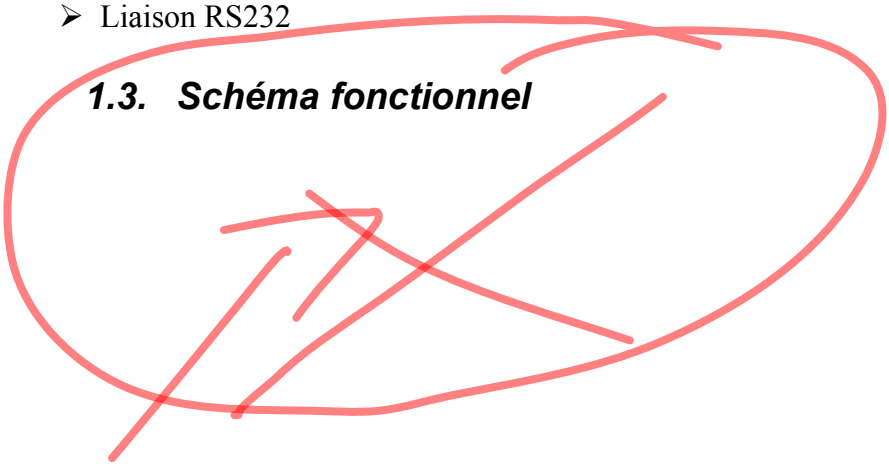
➢ Écran LCD

➢ Logiciel de supervision : C++ Builder 6

➢ Logiciel de programmation d'ATméga : CodeVision AVR

➢ Liaison RS232

## 1.3. Schéma fonctionnel



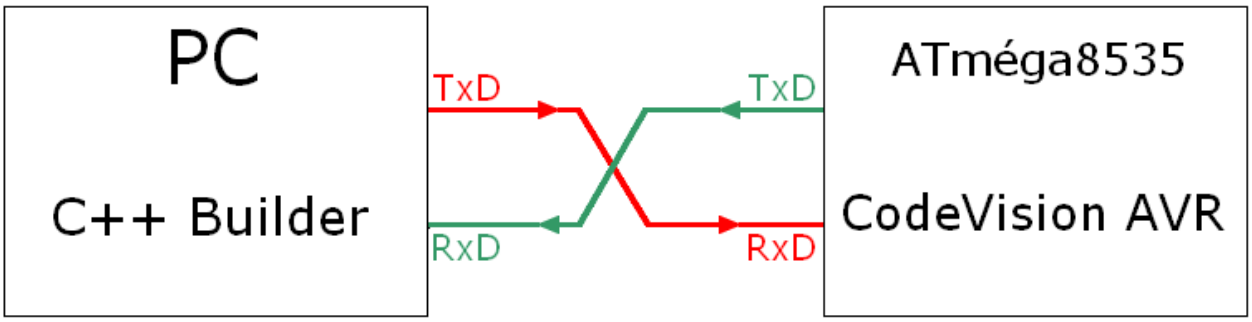



*Illustration 3: Schéma fonctionnel*

Les données sont transmises en série de la sortie TxD vers l'entrée RxD.

On envoie les données sur la sortie TxD du PC vers la carte, composée du microcontrôleur ATméga8535. Celui-ci va recevoir les données sur son entrée RxD puis va émettre une réponse vers l'ordinateur (TxD de l'ATméga8535 vers RxD du PC).

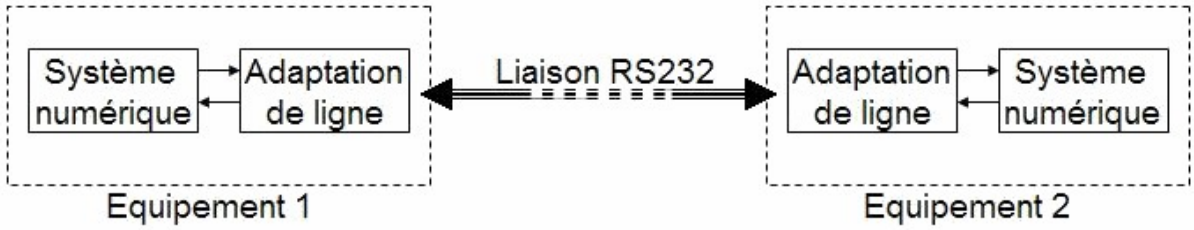



*Illustration 4: Schéma fonctionnel de la liaison RS232*

On peut répartir les fonctionnalités de la manière suivante :

le système numérique, qui est constitué :

- d'un circuit numérique de type microprocesseur. Son rôle est de stocker ou d'envoyer les données sous forme brute et de gérer le contrôle de flux.
- D'un circuit de reconstitution des trames RS232 dont le rôle est de sérialiser les données à envoyer en respectant le protocole RS232.

Le circuit d'adaptation de ligne. Son rôle est de convertir la trame numérique (données sérialisés) en trame électrique, en respectant les niveaux de tension et de courant imposés par la norme du support physique de transmission choisi.

La liaison entre les deux systèmes est constituée de fils transportant les signaux électriques formant les trames.

Il y a le système numérique, un circuit de reconstitution, un circuit d'adaptation, des fonctionnalités...????
MeP doit permettre de reconnaître organisation, structure - ce n est pas le cas ici

## 1.4. Planning

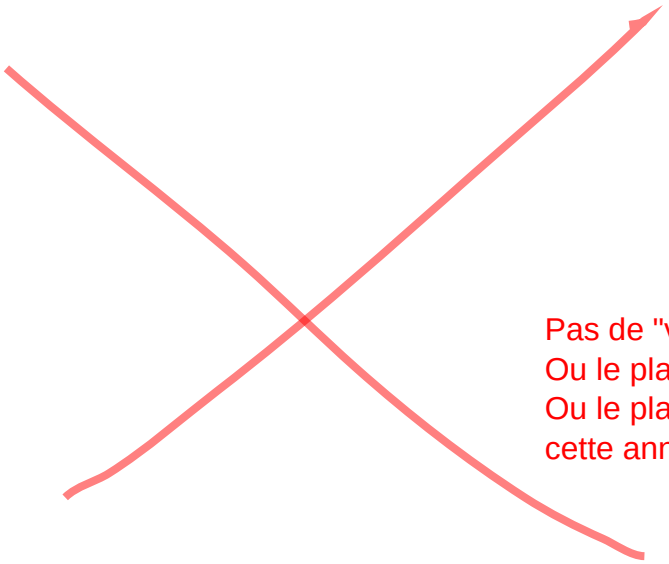
Pour réaliser notre projet dans les temps, nous avons élaboré un planning assez simple, structuré en diverses étapes. Nous devions mettre en service notre projet deux séances avant la fin.

Constat ou contrainte ?

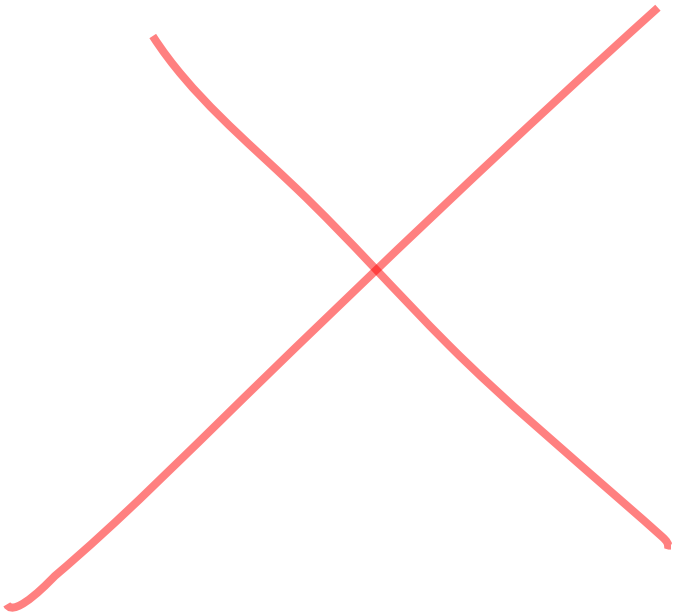
Tout au long de notre projet, nous avons réalisé les tests des programmes parties par parties afin de vérifier le bon fonctionnement de chacune.

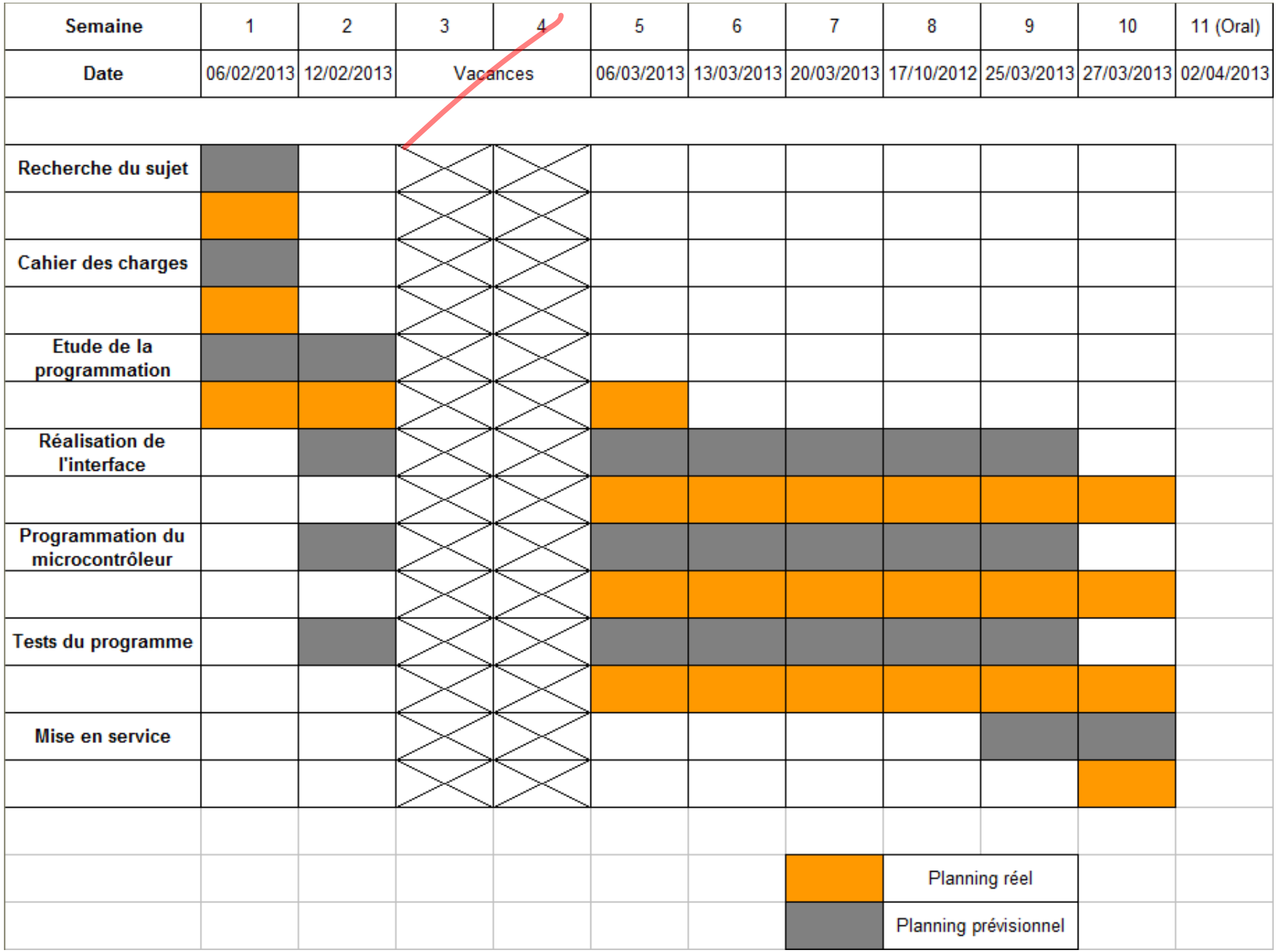
Cependant, si nous faisons un bilan de fin de projet, nous constatons qu'avec les difficultés rencontrées nous avons perdu du temps. La programmation du microcontrôleur s'est révélée plus compliquée que prévu, ainsi que celle de l'interface.

Par rapport au préviosionnel ( d'où son nom // réèl)



Pas de "vide"
Ou le planning "tombe" juste
Ou le placer en annexe et faire un renvoi vers cette annexe




Semaine
1
2
3
4
5
6
7
8
9
10
11 (Oral)
Date
06/02/2013
12/02/2013
Vacances
06/03/2013
13/03/2013
20/03/2013
17/10/2012
25/03/2013
27/03/2013
02/04/2013
Recherche du sujet
Cahier des charges
Etude de la programmation
Réalisation de l'interface
Programmation du microcontrôleur
Tests du programme
Mise en service
Planning réel
Planning prévisionnel

Introduire cette partie
Où en est-on dans la rédaction ? plan de ce qui va suivre

# 2. Étude théorique

## 2.1. Format des trames RS232

Afin de pouvoir programmer l'interface, il a fallu se renseigner sur le format des trames RS232. La transmission étant asynchrone, elle s'effectue sans signal d'horloge. Le récepteur (ici, ATméga8535) peut recevoir sans erreur la donnée série à condition :

- de détecter le début de la trame transmise
- de connaître la fréquence de transmission de chaque bit
- de connaître le format précis de la trame

Ainsi, les UART[3] d'émission et de réception doivent être configurés de manière identique et conformément aux options choisies concernant le format des trames RS232 et la vitesse de transmission. Une trame RS232 est constituée des bits suivants :

- 1 bit de START : c'est un '0' logique. Lorsque la ligne est au repos, elle est au niveau logique '1'. L'émission de ce bit permet au récepteur de détecter le début de la transmission d'une trame, et de se synchroniser avec l'émetteur.

- La donnée de 1 à 8 bits suivant les UART. Il faut savoir que le poids faible de la donnée est transmise en premier.

Gras

- 1 bit de parité (optionnel) : il permet la détection d'une éventuelle erreur de transmission due à un support défaillant, ou à une perturbation électromagnétique. Le calcul du bit de parité est réalisé par l'UART. On peut distinguer deux type de parité :
  - ◆ La parité paire : le nombre de '1' contenus dans l'ensemble donnée (cases de 0 à 7) et parité (case « P ») doit être un nombre pair.
  - ◆ La parité impaire : le nombre de '1' contenus dans l'ensemble donnée et parité doit être un nombre impair.

- 1 ou 1,5 ou 2 bits de STOP : c'est un '1' logique transmis pendant une durée de 1 ; 1,5 ou 2 cycles de transmission. Il permet de maintenir la ligne au repos avant la transmission éventuelle d'une nouvelle trame.

3 UART : Universal Asynchronous Receiver Transmitter

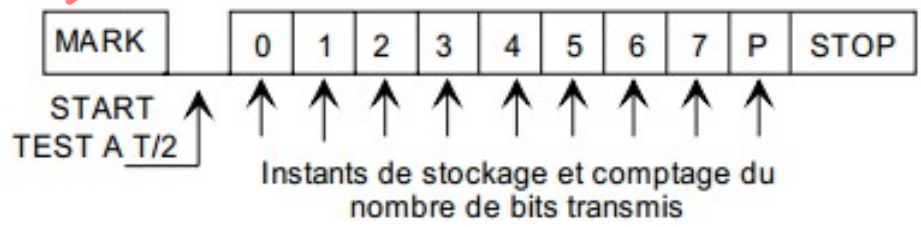



*Illustration 5: Trame RS232*

Voici un exemple : on veut transmettre la donnée 45 en hexadécimal, avec les conditions suivantes : donnée de 7 bits, parité paire, 2 bits de STOP.

Voir illustr. 6

45 en hexadécimal correspond à 1000101 en binaire. Voici donc le chronogramme de la trame logique :

Utiliser index des illustr.

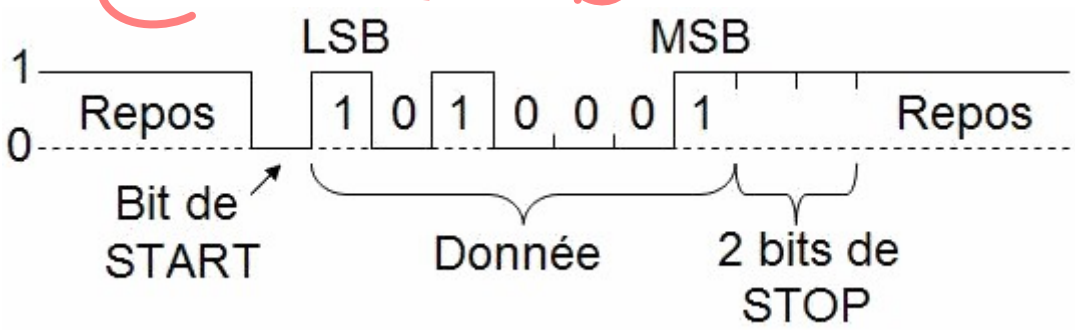



*Illustration 6: Trame RS232*

### 2.2. Carte programmable

Dans le cadre de notre projet nous avons utilisé une carte programmable réalise par M.Lequeu. Elle est composée de 2 boutons poussoirs, d'un afficheur LCD, et d'un Atmega8535.

Nous utiliserons ces différentes partie dans le programme développé sous AVR. Mais nous n'utiliserons pas toutes les fonctionnalités de l'ATmega8535. Nous nous servirons que de L'USART qui permet l'émission et la réception des trames RS232.

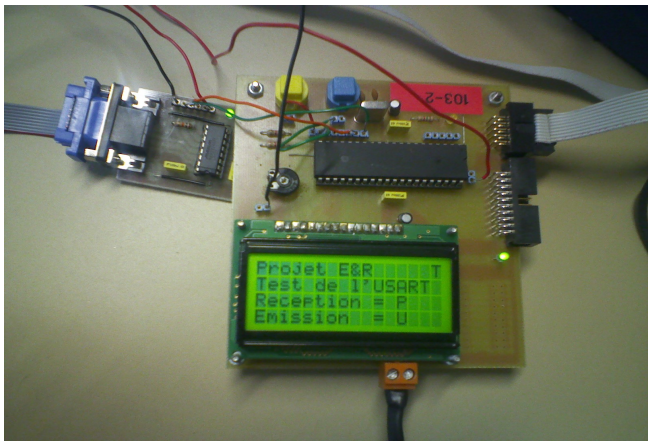



*Illustration 7: carte de programmation et module CON 32*

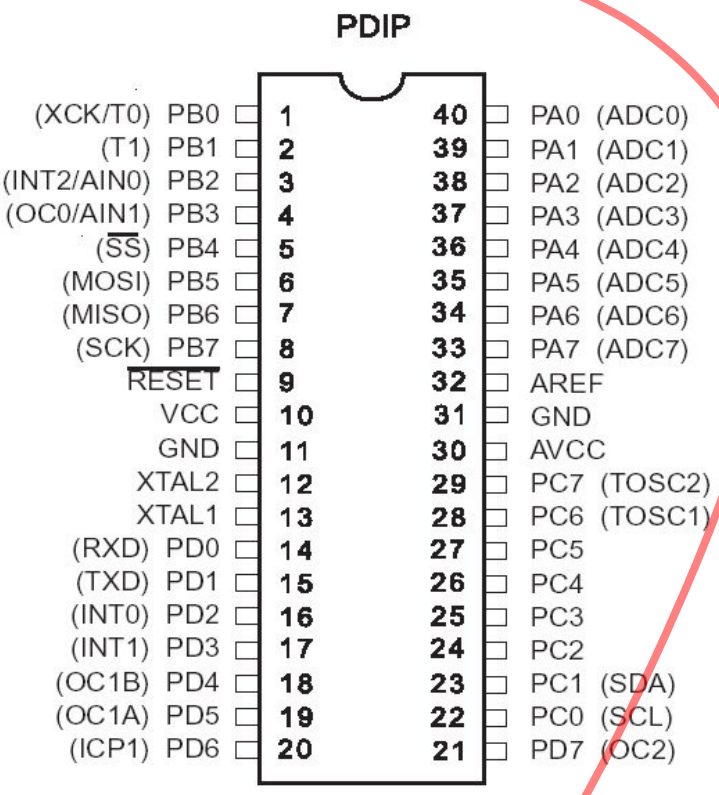



*Illustration 8: Entrées/sorties d'un ATméga8535*

Qu'est-ce que cela vient faire ici ?
N'est pas présenté, commenté ici....?

# 3. Réalisation

Même remarque chap 2

## *3.1. Programme de supervision*

Après s'être renseigné sur le format des trames, nous avons pu commencer la supervision avec C++ Builder.

De plus, pour utiliser la liaison RS232, le composant TComPort était nécessaire pour effectuer cette liaison. C'est ce composant qui va permettre l'envoi et la réception de données.

Nous pouvons voir ci-dessous les protocoles d'ouverture et de fermeture de la liaison série.

```
//---------------------------------------------------------------------------


void __fastcall TForm1::bt_onClick(TObject *Sender)
{
//Ouverture du port (prêt à communiquer)
ComPort1->Connected=true;       // Autre méthode : ComPort1->Open();

}
//---------------------------------------------------------------------------
void __fastcall TForm1::bt_offClick(TObject *Sender)
{
//Fermeture du port (arrêt des communications)

ComPort1->Connected=false;       //Autre méthode : ComPort1->Close();

}
//---------------------------------------------------------------------------
```

*Illustration 9: Extrait du programme (ouverture et fermeture du port)*

Illustr.9 ou 10 ?

Ici, il s'agit du code pour envoyer un texte vers le périphérique qui est connecté sur le port COM1. La propriété « WriteStr » va envoyer le texte écrit sur l'interface vers l'écran LCD via la liaison série.

```
void __fastcall TForm1::bt_envoiClick(TObject *Sender)
{
//Envoi de caractères ASCII vers le périphérique
AnsiString Phrase;
Phrase = Edit1->Text;

if(ComPort1->Connected == 1)
        {
        ComPort1->WriteStr(Phrase); //Écrit toute la chaîne "Phrase" sur le port série
        Memo1->Lines->Add("Emission :" +Phrase);
        }
}
//---------------------------------------------------------------------------
```

*Illustration 10: Code pour envoyer des caractères*

Pour vérifier si il y a des erreurs pendant la communication des données, on va écrire dans l'événement « OnError » du composant TComPort :

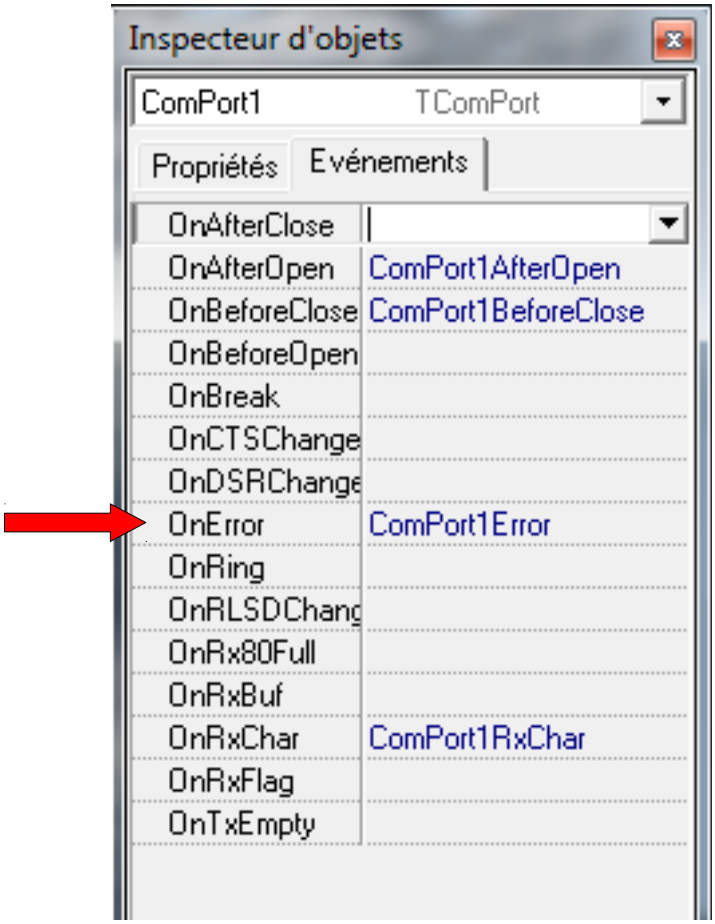



*Illustration 11: Evénements du composant TComPort*

On écrit que si une erreur est détectée, alors je lui demande d'afficher le message « Erreur » sur l'interface :

```
void __fastcall TForm1::ComPort1Error(TObject *Sender, TComErrors Errors)
{
if (Errors.Contains(ceFrame))
    {
        Form1->Memo1->Text = "Erreur"; // erreur de frame (mauvaise vitesse de communication)
    }
if (Errors.Contains(ceOverrun))
    {
        Form1->Memo1->Text = "Erreur"; // erreur d'overrun (lecture du caractère impossible)
    }
}
//---------------------------------------------------------------------------
```

*Illustration 12: Code en cas d'erreur*

Illustr.12

L'extrait de code suivant est écrit dans l'événement OnRxChar du composant TComPort. Il permet de recevoir des chaînes de caractères ou des valeurs. La propriété « ReadStr » lit puis efface un par un les caractères présents dans le buffer d'entrée. Puis j'affiche le résultat dans « Memo1 ».

```
void __fastcall TForm1::ComPort1RxChar(TObject *Sender, int Count)
{
//Lecture d'une chaîne de caractères sur RxD
AnsiString Phrase2;
ComPort1->ReadStr(Phrase2, Count);
    //Lit les "Count" octet(s) présent(s) dans le buffer d'entrée et le(s) place dans Phrase2

Memo1->Lines->Add("Réception :" +Phrase2);


//Lecture d'une valeur sur RxD
unsigned char *Buf = new unsigned char [Count];
    ComPort1->Read(Buf, Count);
    //Lit "Count" octet(s) présent(s) dans le buffer d'entrée et le(s) place dans "Buf"
    delete [] Buf;
    Buf = NULL;
    Memo1->Lines->Add("Réception :" +*Buf);

}
//---------------------------------------------------------------------------
```

*Illustration 13: Code pour lire les données en réception*

Introduire l'illustration 14. Mettre une légende, ce que va faire l'utilisateur... Comme c'est simple alors que....

Voici une photo de l'interface :

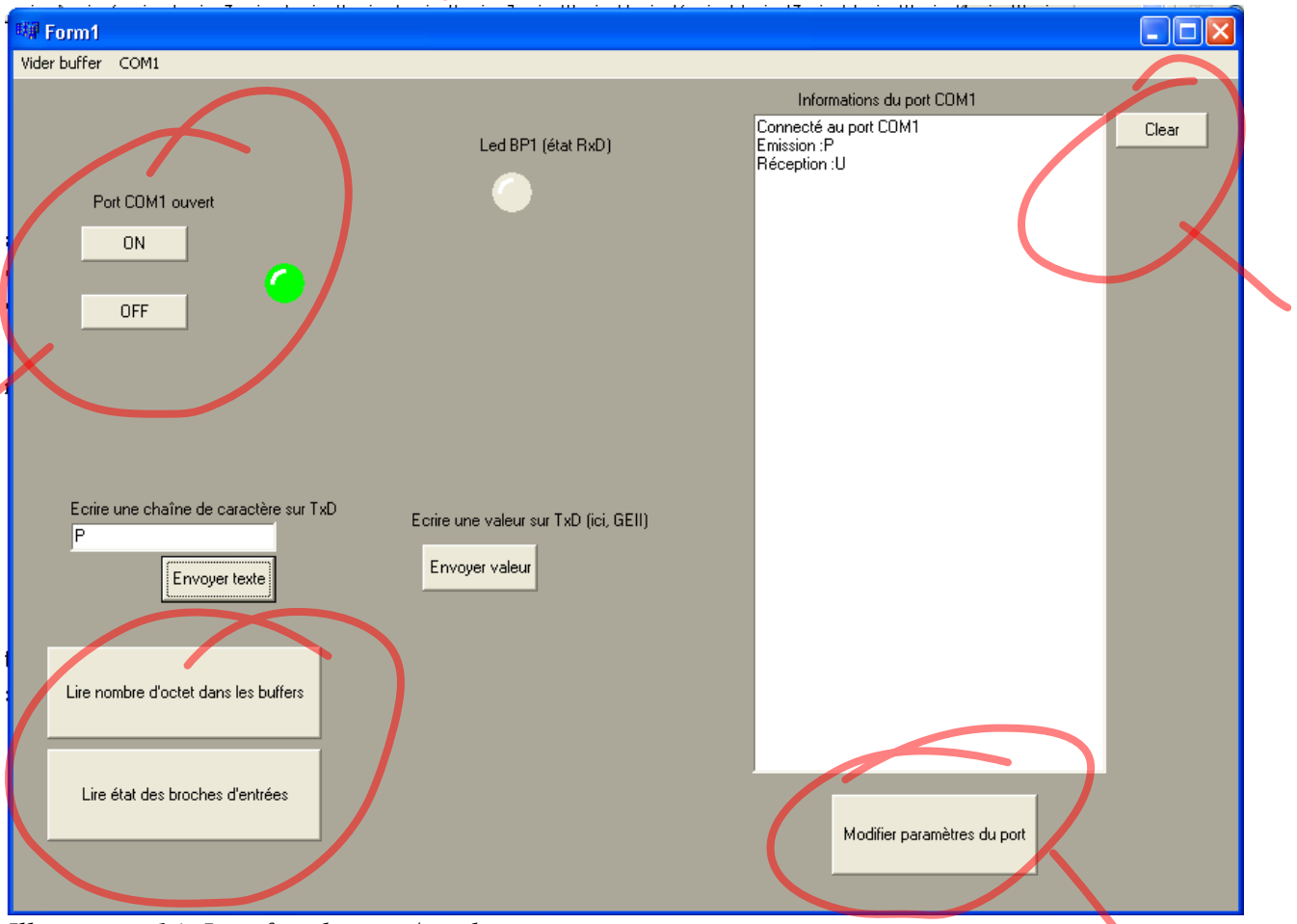



*Illustration 14: Interface homme/machine*

Mettre un bout du code en exemple et le reste en annexe avec renvoi vers celle-ci

### 3.2. *Programme du microcontrôleur*

Le but de ce code est de géré, les boutons poussoirs l'afficheur LCD. De plus il doit pouvoir géré réception de trame RS232 provenant de l'ordinateur, les lires et les afficher sur le LCD. Enfin il doit pouvoir génère des trames RS232 vers l'ordinateur pour lui transmettre des valeurs.

```
Chip type           : Atmega8535
Program type        : Application
Clock frequency     : 16,000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 128
```

Il s'agit du microcontroleur utilisé.
Le type de programme crée est une application.
L'horloge (donc le quartz de la carte) est de 16 MHz

Mettre en vert - ou en tout cas autre police, quelque chose pour faire différence / sur 2 colonnes ?

```
#include <mega8535.h>
```

introduction de la bibliothèque qui permet l'utilisation de l'ATmega8535

introcduction de la biliotheque du LCD, déclaration du port auqu'elle il est ratahé PORTC

```
// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>

// Standard Input/Output functions
#include <stdio.h>
#include <delay.h>
```

Commentaires - autre style

Déclaration de la bibliothèque permettant l'utilisation
Déclaration de la bibliothèque permettant l'utilisation des délais

```
// Declare your global variables here

#define BP1 PIND.7
#define BP2 PIND.6
unsigned char var;
unsigned char tampon[20], test,var2;

void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (0x20)) )
;
/* Put data into buffer, sends the data */
UDR = data;
}

unsigned char USART_Receive( void )
{
/* Wait for data to be received */
```

Le bouton poussoir 1 est relié à la patte 7 du port D
Le bouton poussoir 2 est relié à la patte 8 du port D

Déclaration des variables utilisées dans le programme

initialisation de la capacité à transmettre en RS232

Test de UDRE bit 5 pour voir si l'on transmet

on mes la valeur reçue dans le registre UDR

Initialisation de la réception

```
while ( !(UCSRA & 0x80) )                    Test de RXC bit7 en attente de réception
;
/* Get and return received data from buffer */
return UDR;                                  on retourne la valeur reçue
}


void main(void)
{

unsigned char tampon[20];                    déclaration de variable utilisé que dans le main


PORTA=0x00;      initialisation du PORTA
DDRA=0x00;       initialisation du PORTA en entré sur toute ces broches

PORTB=0x00;      initialisation du PORTB
DDRB=0x00;       initialisation du PORTB en entré

PORTC=0x00;      initialisation du PORTC
DDRC=0x00;       initialisation du PORTC en entré

PORTD=0x00;      initialisation du PORTD
DDRD=0x02;       initialisation du PORTD en entré sauf pour la pin PD1 number 15 TXD qui est
                 en sortie.


// Initialisation de l'USART
// Paramètre de communication: 8 Data, 1 Stop, No Parity
// USART Réception: On
// USART Transition: On
// USART Mode: Asynchrone
// USART Baud : 9600
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x67;

// LCD module initialisation
lcd_init(16);
lcd_gotoxy(0,0);                 ligne 0 colonne 0
lcd_putsf("Projet E&R");         écrire « Projet E&R »
lcd_gotoxy(0,1);                 Ligne 1 colonne 0
lcd_putsf("Test de l'USART");    écrire « Test de l'USART »
var=0x55;                        initialisation de la variable var a la valeur 'U'
```

```
while (1)          boucle qui contient le programme
{
    PORTD.3=1;

    USART_Transmit(var);              Émission sur la liaison série

    var2=USART_Receive();             Réception sur la liaison série d'une valeur mise dans var2

    if (var2 != 0)                                Si l'on reçois quelque chose
    {
     sprintf(tampon,"Reception = %c", var2);     On l'affiche
     lcd_gotoxy(0,2);                            Ligne 2 colonne 0
     lcd_puts(tampon);
     if (var2 == 'B')                            si la variable reçue est 'B'
     {
      var='A';                                   on charge la valeur 'A' dans la zone d'émission
      sprintf(tampon,"Emission  = %1c",var);     on l'affiche
      lcd_gotoxy(0,3);                           Ligne 3 colonne 0
      lcd_puts(tampon);

     }
    }

     var='U';                        On émet 'U' en continue
     if (BP1 == 0)                   Si on appuis sur le bouton poussoir 1
     {
          var='G';                   On émet 'G'
     }

     if (BP2 == 0)                   Si on appuis sur le bouton poussoir 2
     {
          var='S';                   On émet 'S'
     }

     sprintf(tampon,"Emission  = %1c",var);        On affiche la variable var émise
     lcd_gotoxy(0,3);                              Ligne 3 colonne 0
     lcd_puts(tampon);

}
}
```

## Conclusion

L'idée de ce projet nous a été donnée par l'enseignant. Comme le projet était essentiellement basé sur de la programmation, nous avons dû apprendre à utiliser des fonctions spécifiques à la réception et à l'émission de trame pour le bon fonctionnement du projet.

Tout en ayant compris précisément l'aspect théorique de ce sujet, la mise en pratique s'est révélée plus difficile que prévu. Ce projet nous a donné l'occasion de mettre en application différentes connaissances informatiques et de les développer.

Au final, seules quelques parties du projet fonctionnent. Au début nous devions voir l'action des boutons poussoirs sur l'interface via des LED. C'est à dire qu'un appui sur le bouton poussoir devait allumer une LED sur l'écran de l'ordinateur. De plus, nous voulions envoyer une chaîne de caractères sur l'écran LCD et malheureusement nous ne pouvons en afficher qu'un.

## Résumé

Actuellement, le protocole RS232 est de plus en plus remplacé par de l'USB. Mais cette technologie permet de communiquer facilement entre deux systèmes.

Nous avons décidé pour notre projet d'études et réalisations du semestre 4, de réaliser un programme afin de gérer les entrées/sorties d'un microcontrôleur ATméga8535 et ainsi pouvoir allumer une LED ou écrire sur un écran LCD directement depuis une interface située sur l'ordinateur.

Un peu court !

# Bibliographie

[1] **Charente@ctive.fr**. *La liaison RS232*, 2009, [En ligne]. (Page consultée le 10/03/13) <http://montageselectro.charenteactive.fr/Montagec14.aspx>
[2] **Atmel**. *ATméga8535*, 10/2006, [En ligne]. (Page consultée le 10/03/2013) <www.atmel.com/images/doc2502.pdf>
[3] **Vincent PETIT, Patrick PETIT**. *Le composant TComPort, description*, 25/11/2003, [En ligne]. (Page consultée le 10/03/2013) <http://petit.developpez.com/serie/cours_tcomport/#LII-A>
[4] **Thierry LEQUEU**. *ATmega8535 / Programmation de la carte d'étude de l'ATmega8535 - Langage C et assembleur*, 2007, [En ligne]. (Page consultée le 06/02/2013) <http://www.thierry-lequeu.fr/data/DIV517.HTM#02%20-%20Programme%20de%20test%20de%20l%27afficheur%20LCD>
[5] **Lycée Théodore DECK**. *Liaison RS232*, 2012, [En ligne]. (Page consultée le 06/03/2013) <http://www.siloged.fr/cours/html/isn_reseaux_info/la_liaison_srie.html>

[1][2][3][4][5]

# Index lexical

# Index des illustrations

Mettre une page de titre ANNEXES
Avec le sommaire des annexes
Sur chacune mettre index et titre (en style de titreAnnexe)
Ajouter sur chaque page des annexes, quand elles en ont plusieurs, un tête avec un champ titreAnnexe

# Annexe 1

## Programme C++ Builder (interface homme/machine)

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Liaison_serie_app.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "_GClass"
#pragma link "AbLED"
#pragma link "CPort"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------


void __fastcall TForm1::bt_onClick(TObject *Sender)
{
//Ouverture du port (prêt à communiquer)
ComPort1->Connected=true;      // Autre méthode : ComPort1->Open();

}
//---------------------------------------------------------------------------
void __fastcall TForm1::bt_offClick(TObject *Sender)
{
//Fermeture du port (arrêt des communications)

ComPort1->Connected=false;      //Autre méthode : ComPort1->Close();

}
//---------------------------------------------------------------------------


void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
//Fermeture du port (arrêt des communications) lorsqu'on ferme l'application
ComPort1->Connected=false;    // Ou : ComPort1->Close();
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ComPort1Error(TObject *Sender, TComErrors Errors)
{
if (Errors.Contains(ceFrame))
   {
      Form1->Memo1->Text = "Erreur"; // erreur de frame (mauvaise vitesse de communication)
   }
if (Errors.Contains(ceOverrun))
   {
      Form1->Memo1->Text = "Erreur"; // erreur d'overrun (lecture du caractère impossible)
   }
}
//---------------------------------------------------------------------------


void __fastcall TForm1::bt_envoiClick(TObject *Sender)
{
//Envoi de caractères ASCII vers le périphérique
AnsiString Phrase;
Phrase = Edit1->Text;

if(ComPort1->Connected == 1)
      {
      ComPort1->WriteStr(Phrase); //Écrit toute la chaîne "Phrase" sur le port série
      Memo1->Lines->Add("Emission :" +Phrase);
      }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::bt_valeurClick(TObject *Sender)
{

unsigned char tableau[4] = {'G', 'E', 'I', 'I'};

if(ComPort1->Connected == 1)
      {
      ComPort1->Write(tableau, 4); //Ecrit 4 octets de "tableau" sur le port série
      Memo1->Lines->Add(("Emission :")+(ComPort1->Write(tableau, 4)));
      }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ComPort1RxChar(TObject *Sender, int Count)
{
//Lecture d'une chaîne de caractères sur RxD
AnsiString Phrase2;
ComPort1->ReadStr(Phrase2, Count);
   //Lit les "Count" octet(s) présent(s) dans le buffer d'entrée et le(s) place dans Phrase2
```

```
Memo1->Lines->Add("Réception :" +Phrase2);


//Lecture d'une valeur sur RxD
unsigned char *Buf = new unsigned char [Count];
   ComPort1->Read(Buf, Count);
   //Lit "Count" octet(s) présent(s) dans le buffer d'entrée et le(s) place dans "Buf"
   delete [] Buf;
   Buf = NULL;
   Memo1->Lines->Add("Réception :" +*Buf);

}
//---------------------------------------------------------------------------

void __fastcall TForm1::bt_bufferClick(TObject *Sender)
{

if(ComPort1->Connected == 1)
      {
      ComPort1->ClearBuffer(true, false); // buffer d'entrée vidé, celui de sortie intacte
      }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::bt_octetClick(TObject *Sender)
{
//Lit le nombre d'octet dans le buffer d'entrée
if(ComPort1->Connected == 1)
      {
      int NbrBits;
      NbrBits = ComPort1->InputCount();
      Memo1->Lines->Add((AnsiString) NbrBits);
      }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::bt_paramClick(TObject *Sender)
{
//Régler les paramètres de la liaison série (vitesse, parité, bit de stop...)
if(ComPort1->Connected == 1)
      {
      ComPort1->ShowSetupDialog ();
      }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ComPort1AfterOpen(TObject *Sender)
{
//A l'ouverture du port COM1
```

```
Form1->LED_verif->Checked=true;
Form1->portcom1ouvert->Visible = true;
Form1->portcom1ferme->Visible = false;

Memo1->Lines->Add("Connecté au port COM1");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ComPort1BeforeClose(TObject *Sender)
{
//A la fermeture du port COM1
Form1->LED_verif->Checked = 0;
Form1->portcom1ouvert->Visible = false;
Form1->portcom1ferme->Visible = true;

Memo1->Lines->Add("Déconnecté du port COM1");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::bt_clearClick(TObject *Sender)
{
//Efface le memo
Memo1->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::desortie1Click(TObject *Sender)
{
if(ComPort1->Connected == 1)
        {
        ComPort1->ClearBuffer(false, true); // buffer d'entrée intact, celui de sortie vidé
        Memo1->Lines->Add("Buffer de sortie vidé.");
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::dentre1Click(TObject *Sender)
{
if(ComPort1->Connected == 1)
        {
        ComPort1->ClearBuffer(true, false); // buffer d'entrée vidé, celui de sortie intact
        Memo1->Lines->Add("Buffer d'entrée vidé.");
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::les21Click(TObject *Sender)
{
if(ComPort1->Connected == 1)
        {
```

```
        ComPort1->ClearBuffer(true, true); // buffers d'entrée et de sortie vidés
        Memo1->Lines->Add("Buffers d'entrée et de sortie vidés.");
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Ouvrir1Click(TObject *Sender)
{
ComPort1->Open();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Fermer1Click(TObject *Sender)
{
ComPort1->Close();
}
//---------------------------------------------------------------------------
```

Nouvelle page, nouvelle annexe

## Annexe 2

Code l'ATmega développer sous avr.

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.3 Evaluation
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
Project :
Version :
Date    : 08/07/2007
Author  : Freeware, for evaluation and non-commercial use only
Company :
Comments:
Chip type           : ATmega8535
Program type        : Application
Clock frequency     : 16,000000 MHz
```

```
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 128
*****************************************************/

#include <mega8535.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>

// Standard Input/Output functions
#include <stdio.h>
#include <delay.h>

// Declare your global variables here

#define BP1 PIND.7
#define BP2 PIND.6
unsigned char var;
unsigned char tampon[20], test,var2;
//unsigned char ,seconde,seconde1,temps,temps1,dizaine,unite,dixieme,centieme;

void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
```

```
while ( !( UCSRA & (0x20)) ) // Test de UDRE bit 5
;
/* Put data into buffer, sends the data */
UDR = data;
}

unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !(UCSRA & 0x80) ) // Test de RXC bit7
;
/* Get and return received data from buffer */
return UDR;
}

/*interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
test=UDR; // Lecture du registre de reception.
if (test == "1")
  {
    sprintf(tampon,"coucou");
      lcd_gotoxy(0,2);
      lcd_puts(tampon);
  }

if (test == "0")
  {
    sprintf(tampon,"sa marche");
```

```
    lcd_gotoxy(0,2);
    lcd_puts(tampon);
  }
} */
void main(void)
{
// Declare your local variables here
unsigned char tampon[20];
// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;
// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=Out Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=0 State0=T
PORTD=0x00;
```

```
DDRD=0x02;  // The pin PD1 number 15 TXD is output.
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
```

```
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
UCSRB=0x18;
```

```
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x67;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
// LCD module initialization
lcd_init(16);
lcd_gotoxy(0,0);
lcd_putsf("Projet E&R");
lcd_gotoxy(0,1);
lcd_putsf("Test de l'USART");
test=0;
// UDR = 0x55; // ???
var=0x55;
// Global enable interrupts
#asm("sei")
while (1)
{
    PORTD.3=1;
    lcd_gotoxy(15,0);
    lcd_putsf("-");
    USART_Transmit(var);        // Emission sur la liaison serie
    lcd_gotoxy(15,0);
    lcd_putsf("T");
    // Reception sur la liaison série :
```

```
var2=USART_Receive();
    lcd_gotoxy(15,0);
    lcd_putsf("R");

    if (var2 != 0)
    {
     sprintf(tampon,"Reception = %c", var2);
     lcd_gotoxy(0,2);
     lcd_puts(tampon);
     if (var2 == 'B')
     {
      var='A';
      sprintf(tampon,"Emission  = %1c",var);
      lcd_gotoxy(0,3);
      lcd_puts(tampon);
     }
    }

     var='U';
     if (BP1 == 0)
     {      // BP1
            var='G';

     }

     if (BP2 == 0)
     {      // BP2
            var='S';
```

```
        }
         sprintf(tampon,"Emission  = %1c",var);
        lcd_gotoxy(0,3);
        lcd_puts(tampon) ;
}
}
```