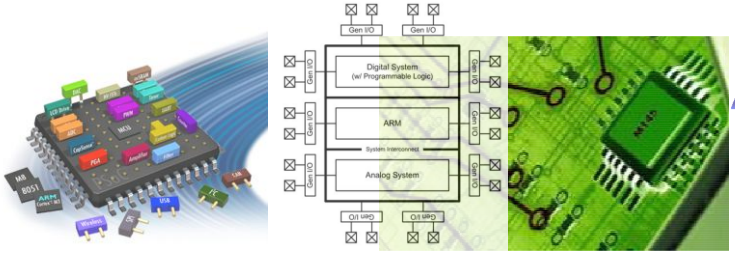
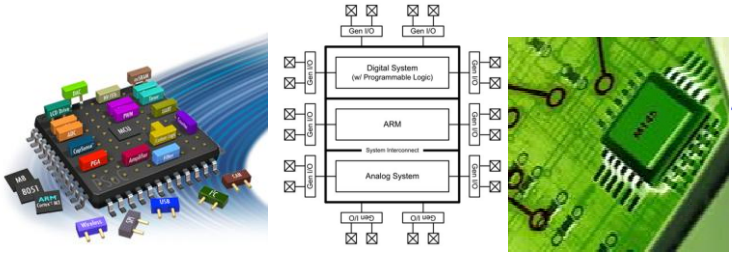


Transmission de données PSoC = Delphi

1 Sommaire :

1 Sommaire : 1
2 Ouverture du projet 2
3 Description de la configuration utilisée 3
4 Mise en œuvre sur la platine de prototypage rapide 4
5 Mise en œuvre logicielle 8
6 L'IHM RS232 départ élève 11
7 Programmation Delphi, ... 13
8 Réception de caractères par le PSoC 15

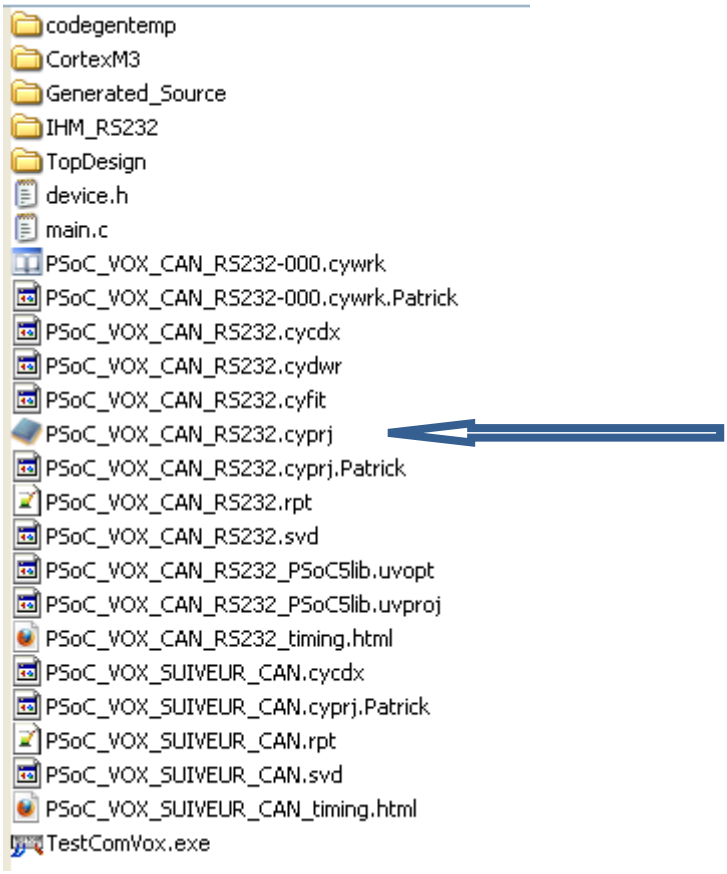


2 Ouverture du projet

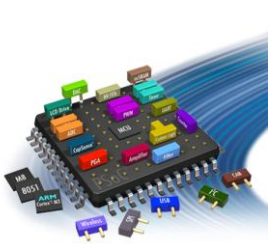
Dans un premier temps nous allons utiliser la platine

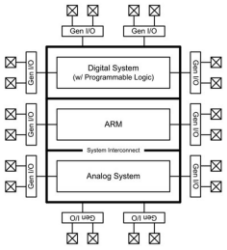
PSoC sans cartes additionnelles. Le projet est stocké dans le répertoire

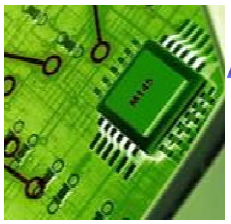
PSoC_VOX_CAN_RS232.cydsn Pour démarrer double cliquer sur le fichier cypress_projet :



Ce projet se propose d'envoyer à une **I**nterface **H**omme **M**achine les données acquises via le projet PSoC CAN. Les solutions mises en œuvre dans ce cadre seront tout à fait adaptables et pourront servir de base de départ pour un échange de données dans les projets de spécialité.



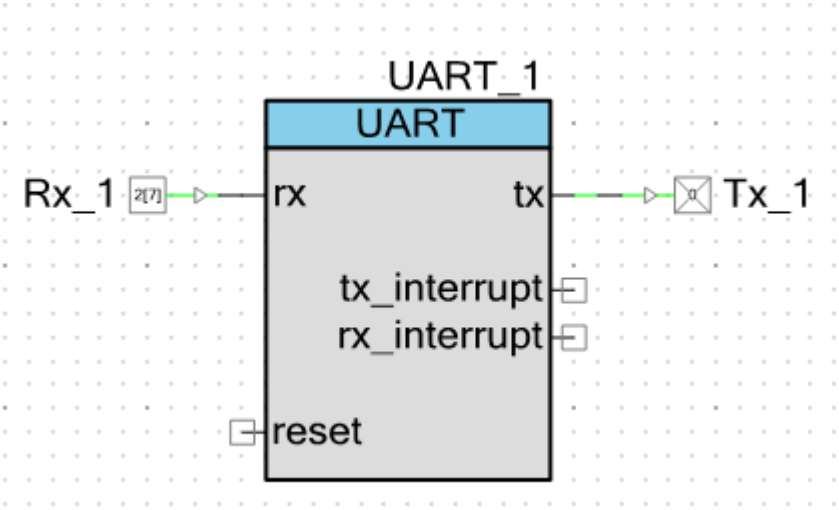


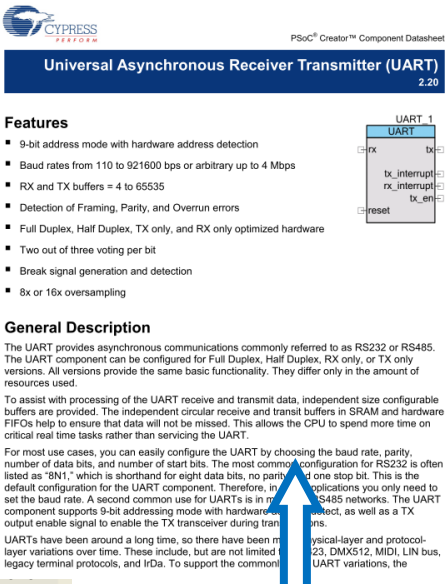


3 Description de la configuration utilisée

Ajout d'un composant liaison série dans un projet :

Il faut ajouter le composant UART dans notre design, puis le configurer :





CYPRESS PERFORM

PSoC® Creator™ Component Datasheet

Universal Asynchronous Receiver Transmitter (UART)
2.20

Features

- 9-bit address mode with hardware address detection
- Baud rates from 110 to 921600 bps or arbitrary up to 4 Mbps
- RX and TX buffers = 4 to 65535
- Detection of Framing, Parity, and Overrun errors
- Full Duplex, Half Duplex, TX only, and RX only optimized hardware
- Two out of three voting per bit
- Break signal generation and detection
- 8x or 16x oversampling

UART_1
UART
rx
tx
tx_interrupt
rx_interrupt
tx_en
reset

General Description

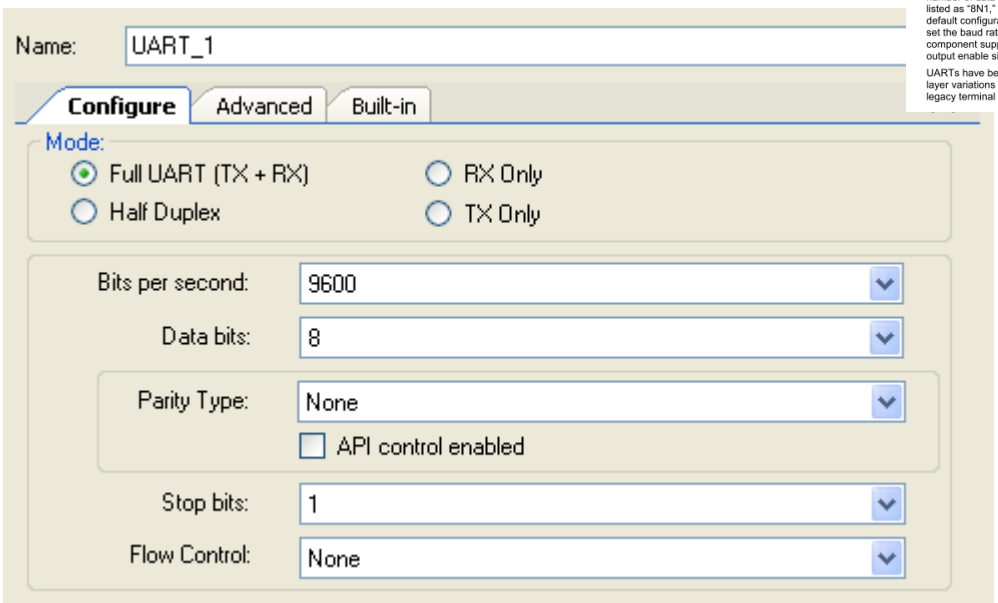
The UART provides asynchronous communications commonly referred to as RS232 or RS485. The UART component can be configured for Full Duplex, Half Duplex, RX only, or TX only versions. All versions provide the same basic functionality. They differ only in the amount of resources used.

To assist with processing of the UART receive and transmit data, independent size configurable buffers are provided. The independent circular receive and transit buffers in SRAM and hardware FIFOs help to ensure that data will not be missed. This allows the CPU to spend more time on critical real time tasks rather than servicing the UART.

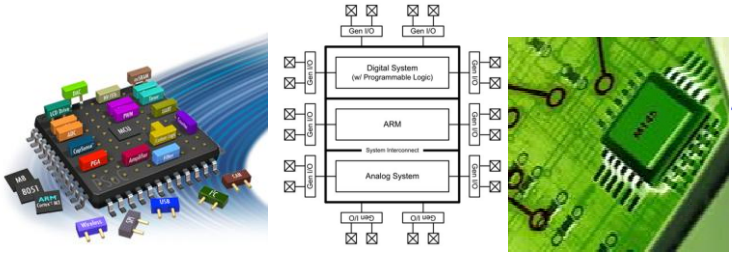
For most use cases, you can easily configure the UART by choosing the baud rate, parity, number of data bits, and number of start bits. The most common configuration for RS232 is often listed as "8N1," which is shorthand for eight data bits, no parity and one stop bit. This is the default configuration for the UART component. Therefore, in many applications you only need to set the baud rate. A second common use for UARTs is in multi-drop RS485 networks. The UART component supports 9-bit addressing mode with hardware address detect, as well as a TX output enable signal to enable the TX transceiver during transmissions.

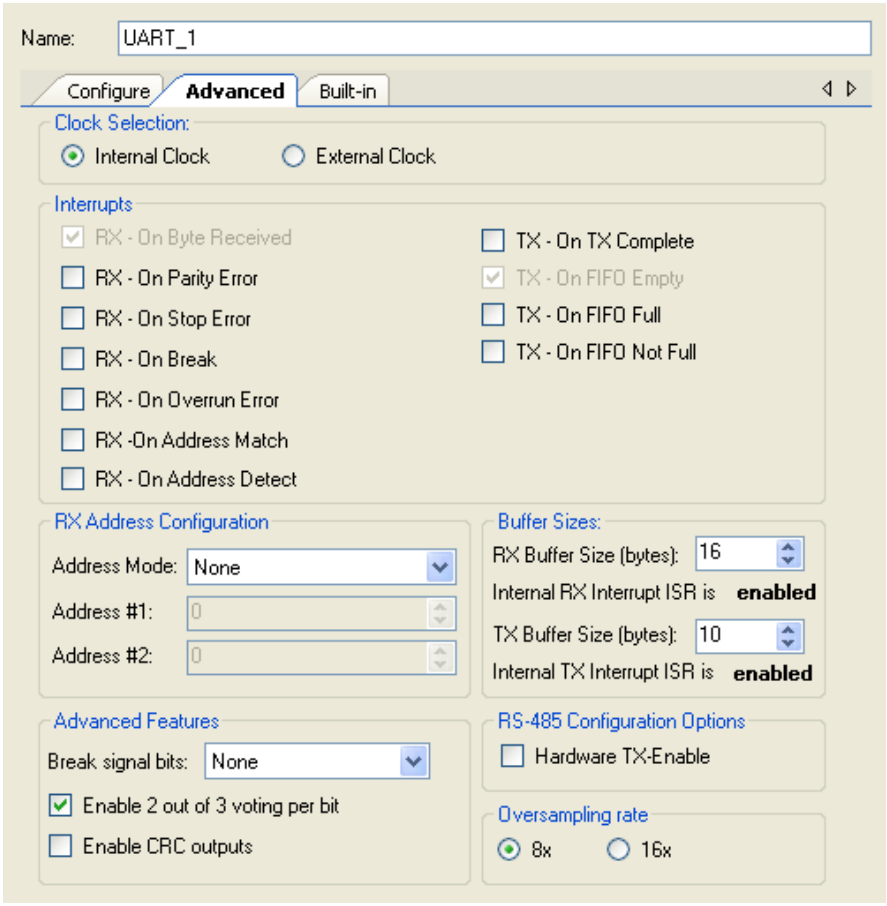
UARTs have been around a long time, so there have been many physical-layer and protocol-layer variations over time. These include, but are not limited to, IrDA, DMX512, MIDI, LIN bus, legacy terminal protocols, and IrDa. To support the common UART variations, the

Configuration du composant :



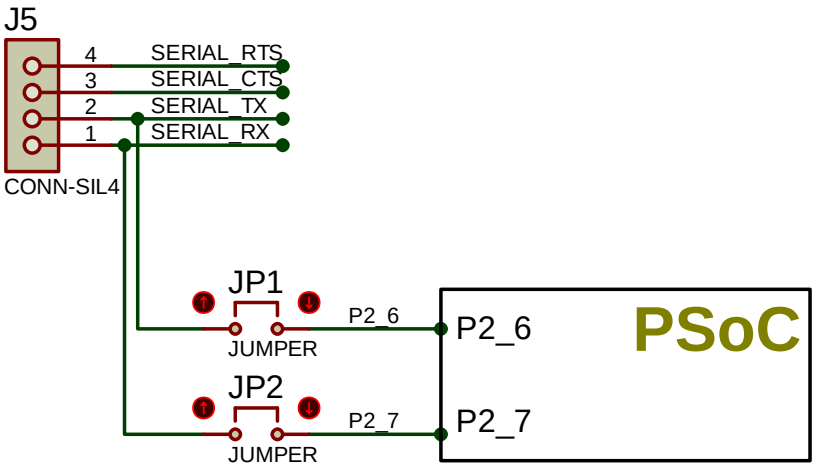
Ne pas oublier les datasheet des composants, pour l'UART 51 pages !

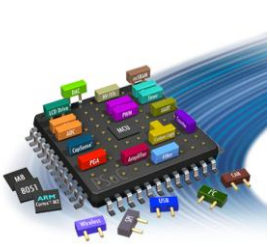


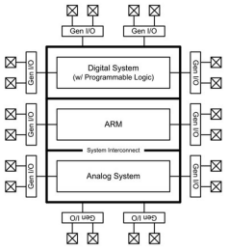


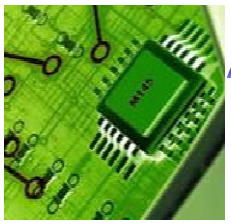
4 Mise en œuvre sur la platine de prototypage rapide

Nous pouvons utiliser l'UART prévue sur la platine PSoCVOX sur les Ports P2_6 et P2_7. Il suffit de positionner les jumpers J1 et J2.

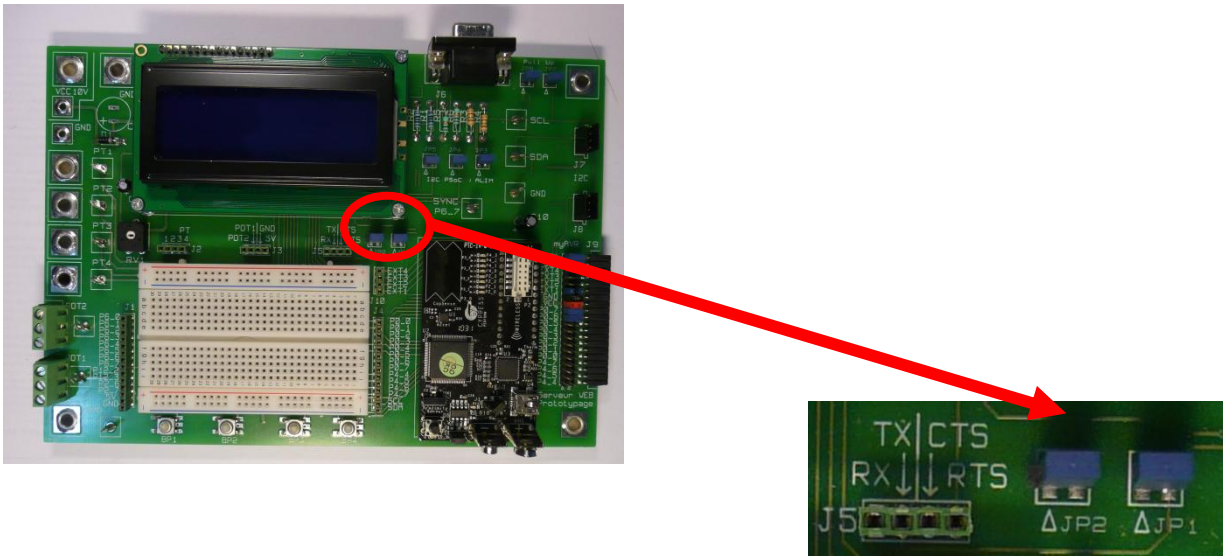


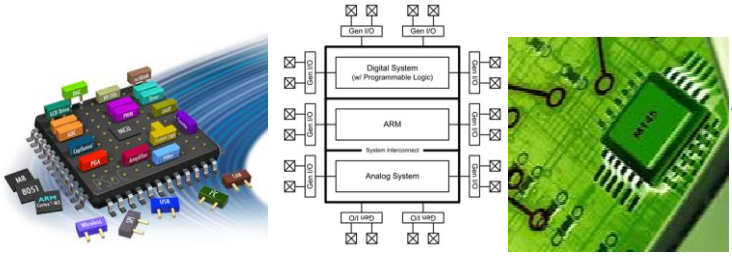






Emplacement des jumpers JP1 JP2 sur la carte PSoCVOX :

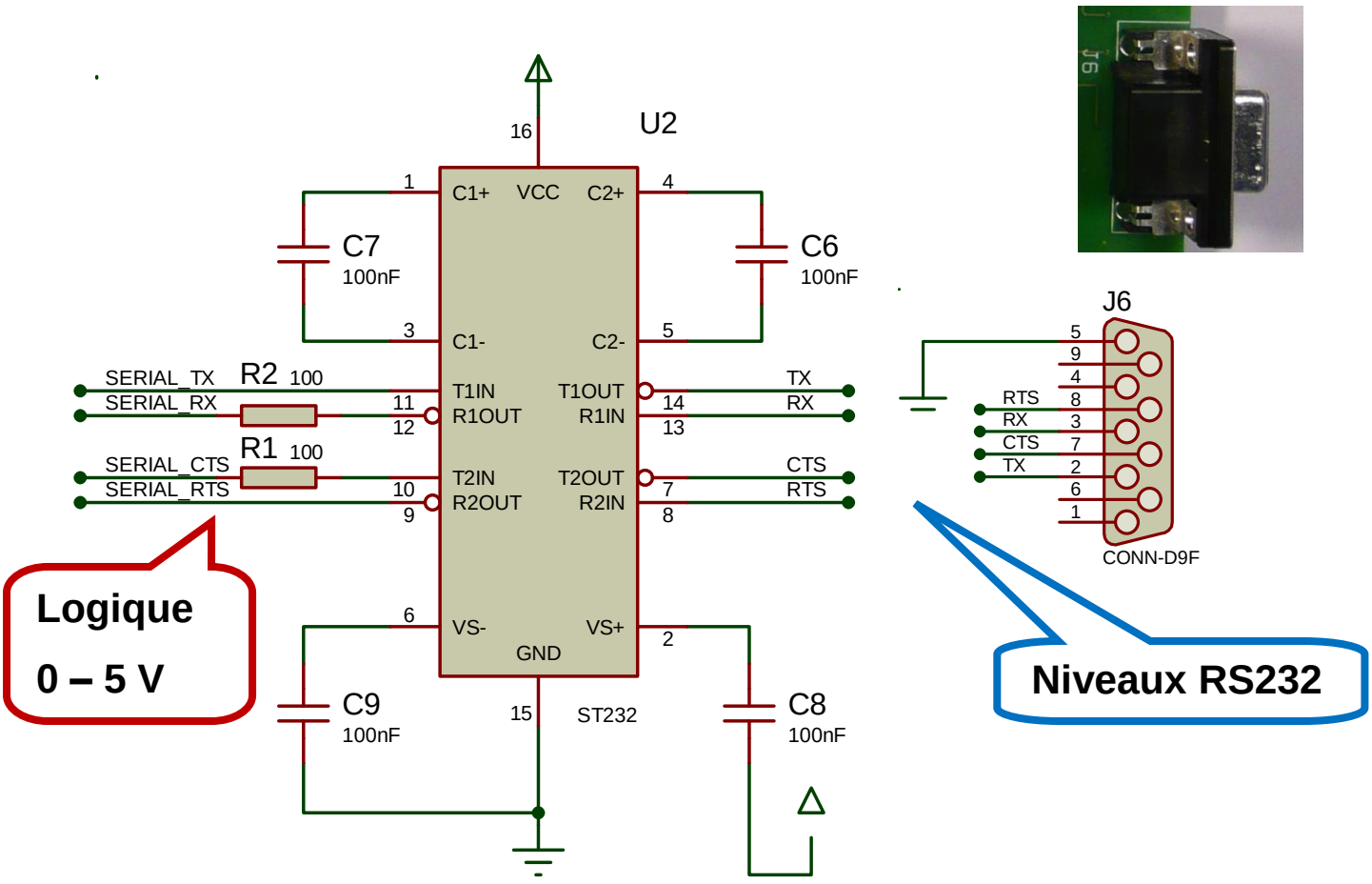




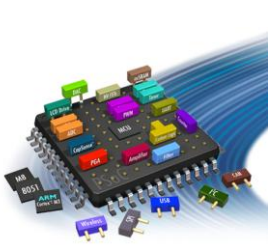
Liaison RS232 un peu d'électronique

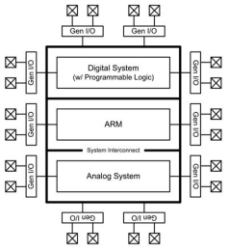
- La liaison série RS232 générée par l'UART dans le micro-contrôleurun signal logique de niveau 0-5V. La normalisation de la liaison RS232 impose que le '0' logique soit représenté par une tension comprise entre +3V...+25V et le '1' logique soit représenté par une tension entre -3V...-25V.

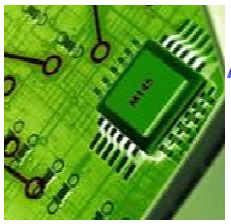
- L'interface doit être bidirectionnelle, le circuit ST232 réalise l'adaptation.



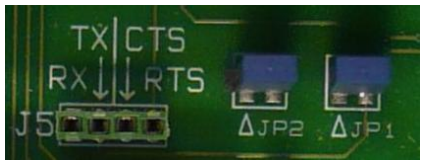
➪ Déterminer à partir de la documentation technique du circuit ST232 le niveau des tensions des signaux RS232,Tx Rx.

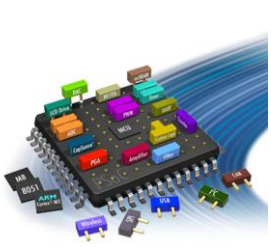


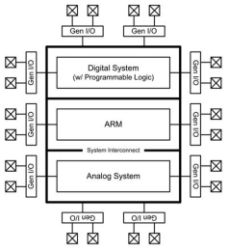


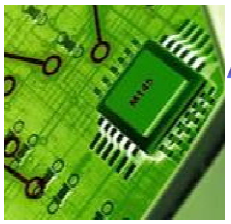


- Noter qu'il est possible d'utiliser le circuit d'interface sur n'importes quelles broches disponibles du PSoC car en enlevant les straps JP1 et JP2 on peut accéder au circuit d'interface par le bornier J5 :









5 Mise en œuvre logicielle

Le composant UART doit être initialisé :

```
// Initialisation UART
UART_1_Start();
```

L'envoi des données par le PSoC se fait très simplement :

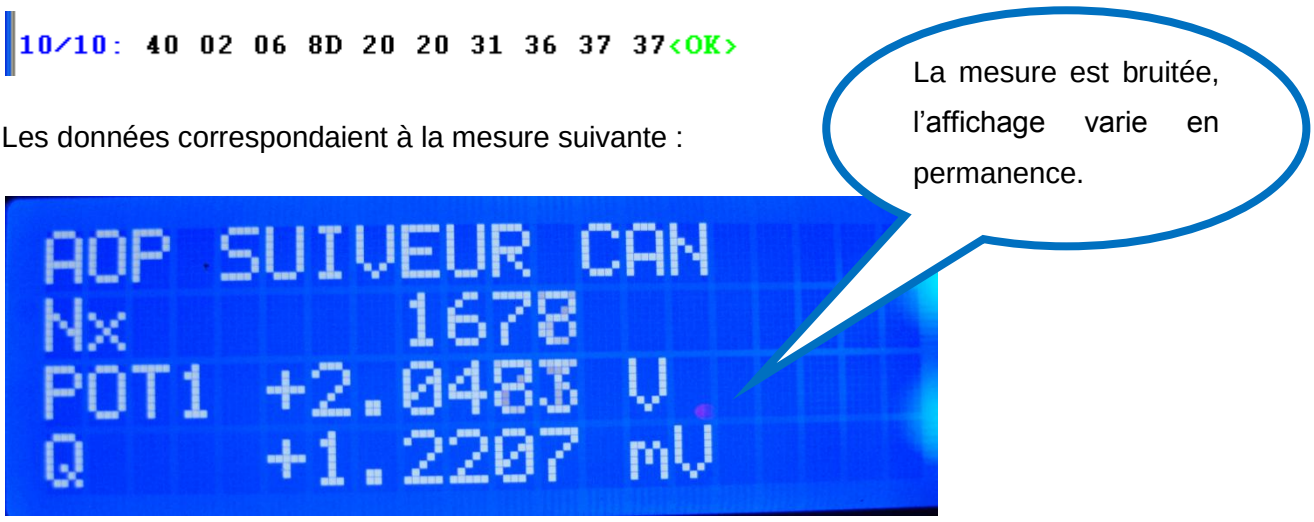
```
UART_1_WriteTxData(0x40);
UART_1_WriteTxData(0x02);
UART_1_WriteTxData(Nx.octet.hi);
UART_1_WriteTxData(Nx.octet.lo);
UART_1_PutString(tstr);
UART_1_WriteTxData(0x0D);
```

Voilà le résultat :

La chaine reçue par le PC saisie au vol avec un sniffer :

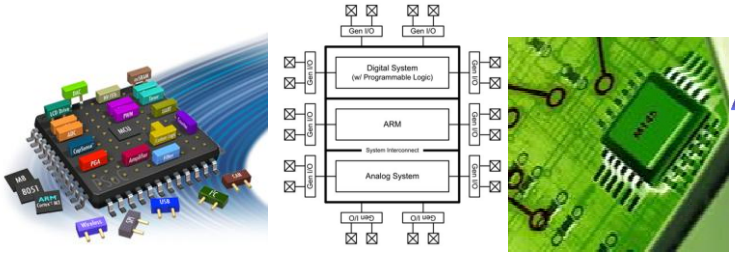
```
10/10: 40 02 06 8D 20 20 31 36 37 37<OK>
```

Les données correspondaient à la mesure suivante :



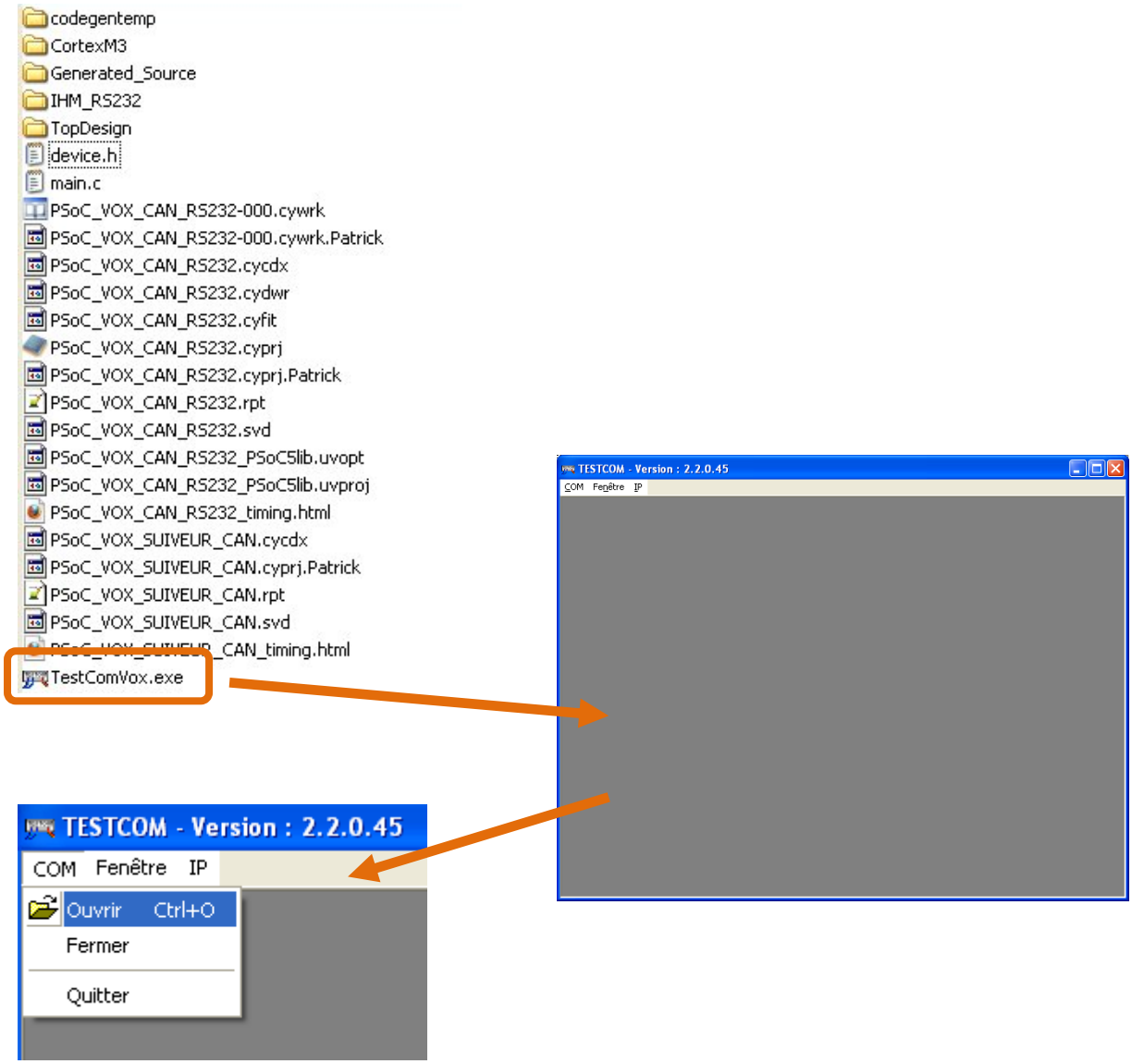
➪ Retrouver la valeur du nombre Nx à partir des données hexadécimales hi.lo

➪ Retrouver la valeur du potentiomètre.

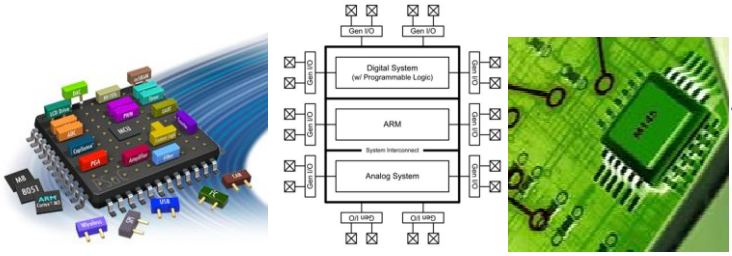


A vous de jouer mise en œuvre du sniffer RS232

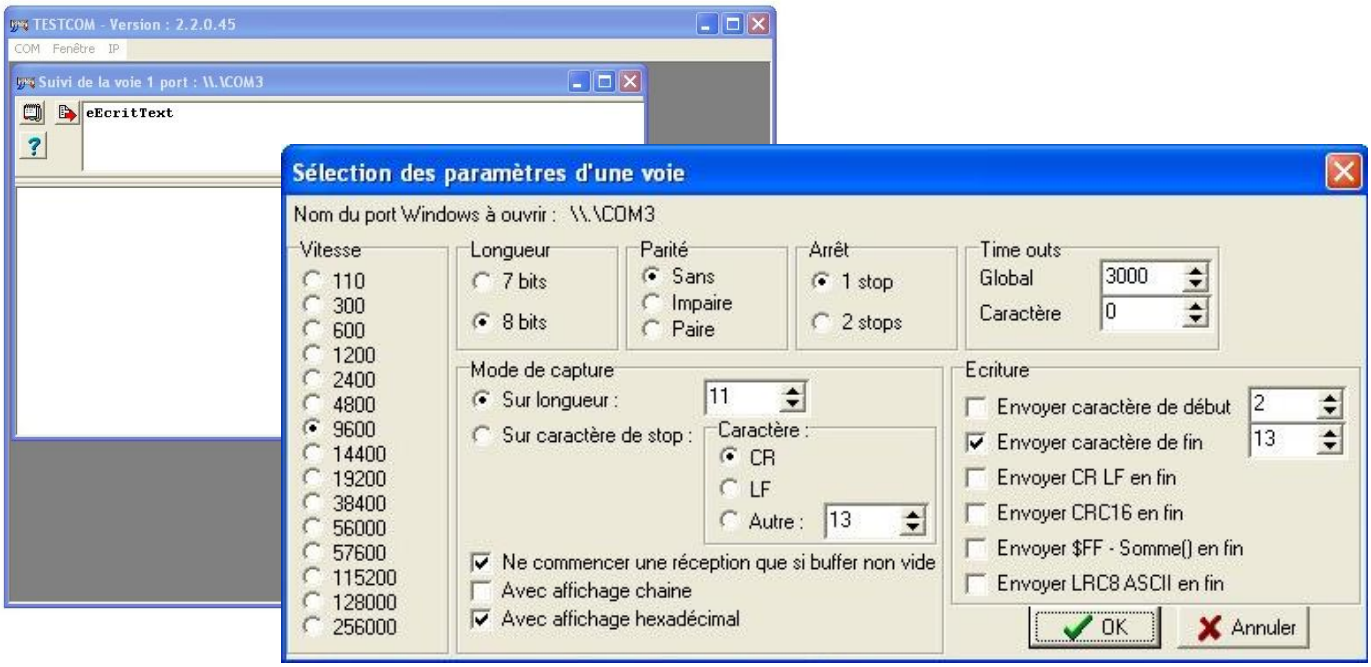
Lancement du programme :



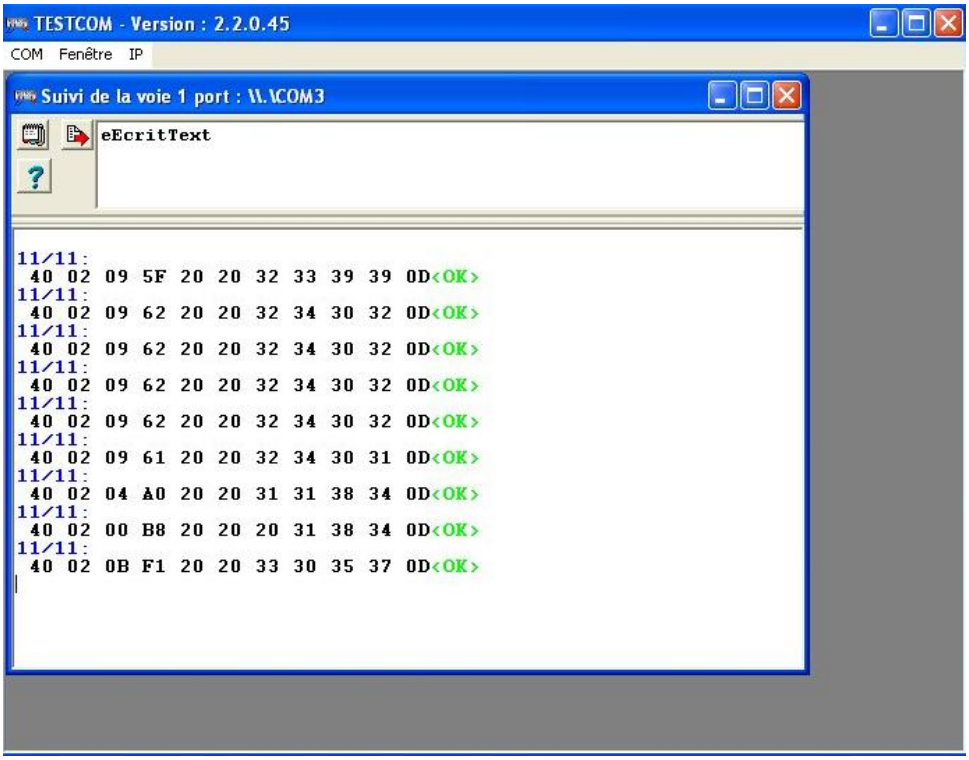
On lance le programme, puis on ouvre le port com avec lequel l'on veut agir.

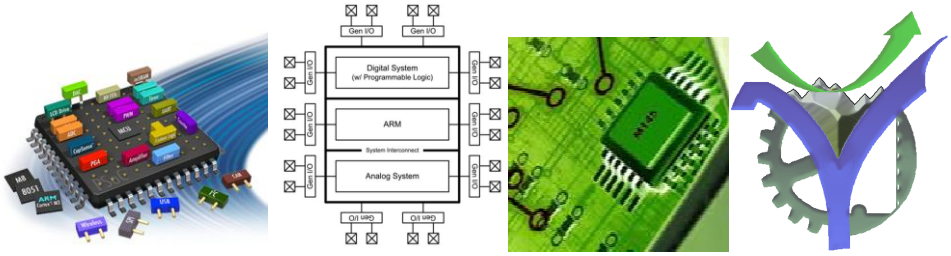


Configuration du fonctionnement en émission et en réception :



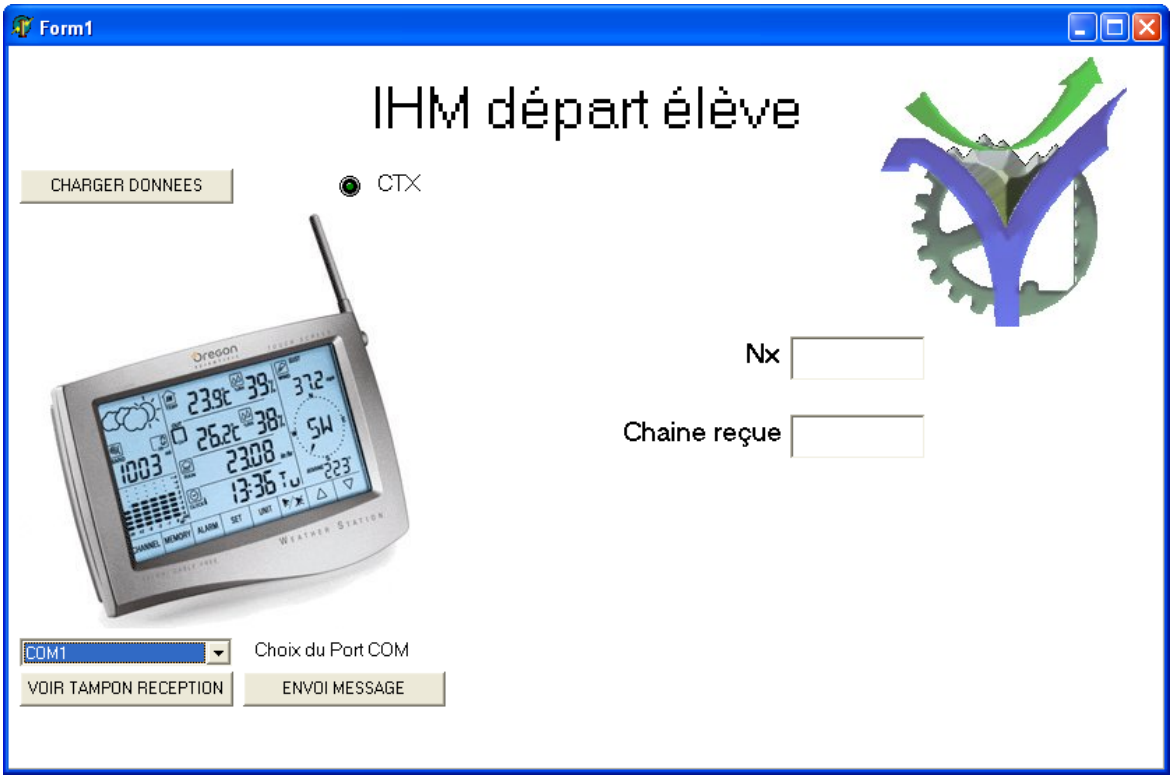
Réception des caractères voilà quelques trames reçues :





6 L'IHM RS232 départ élève

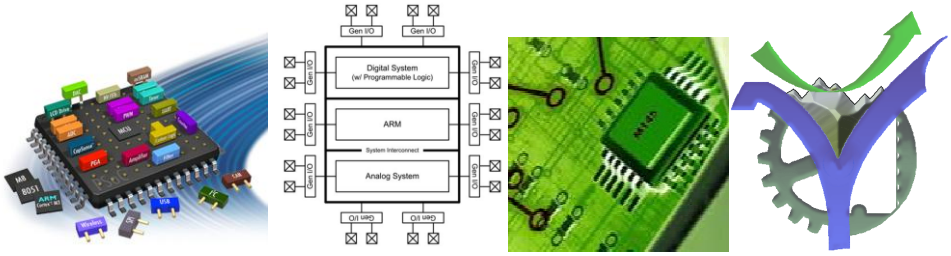
Lancer l'IHM



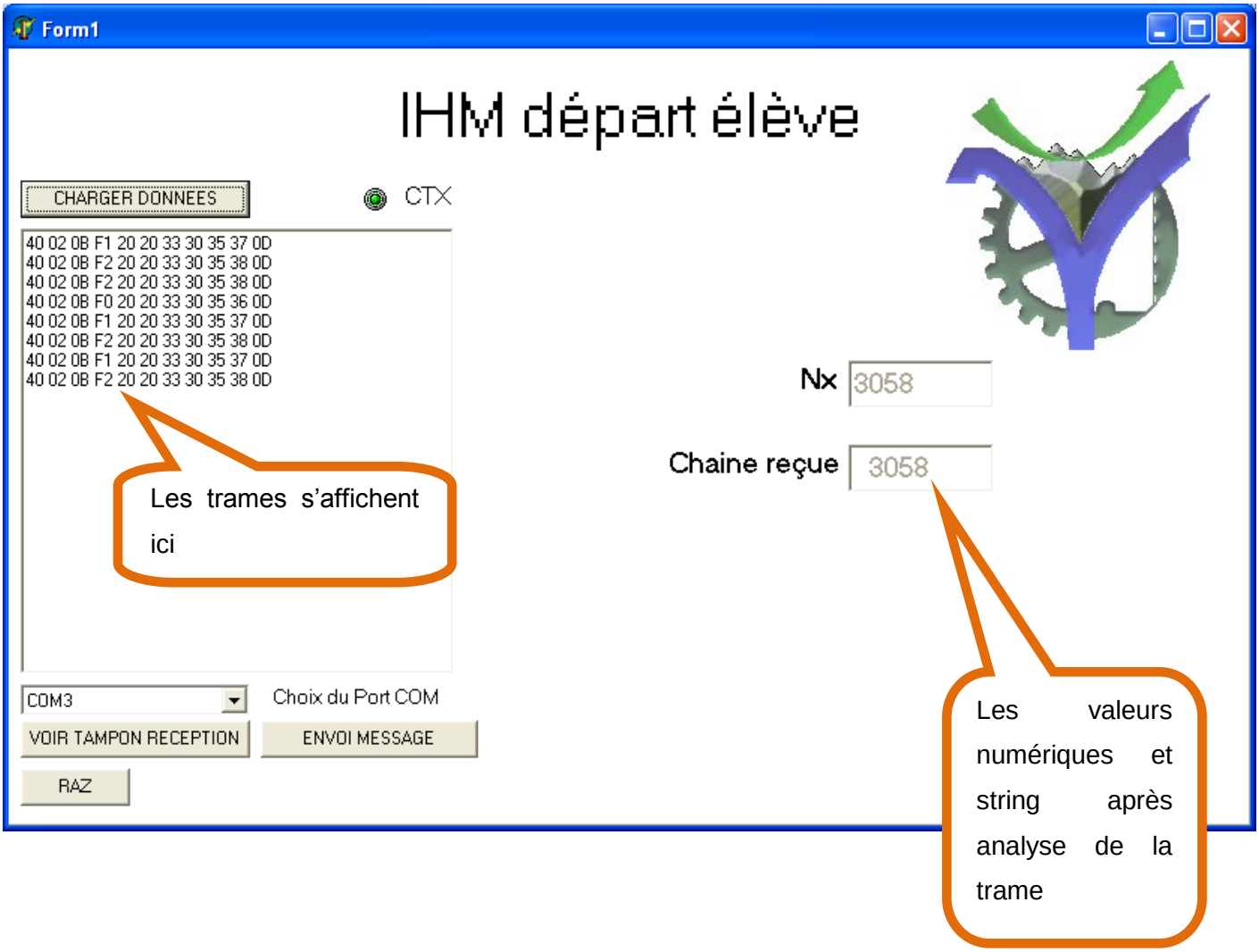
1°. Sélectionner le port com

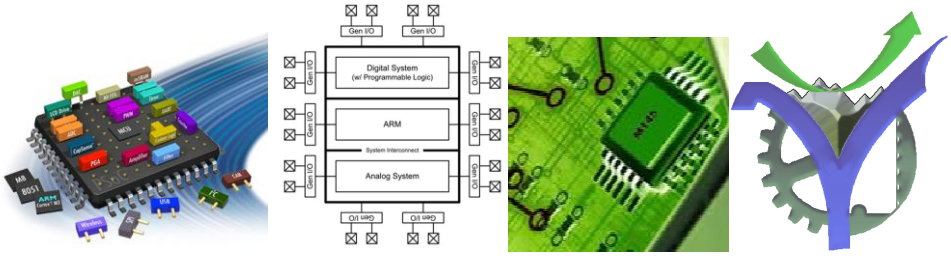
2°. cliquer sur le bouton **voir tampon réception**

3°. puis sur le bouton **charger données**, le contrôle CTX s'allume en vert.



La réception des données se réalise :





7 Programmation Delphi,

Analyse de la trame reçue qui est contenue dans un tableau de caractère de type string.

Rappel du format de la trame :

```
UART_1_WriteTxData(0x40);
UART_1_WriteTxData(0x02);
UART_1_WriteTxData(Nx.octet.hi);
UART_1_WriteTxData(Nx.octet.lo);
UART_1_PutString(tstr);
UART_1_WriteTxData(0x0D);
```

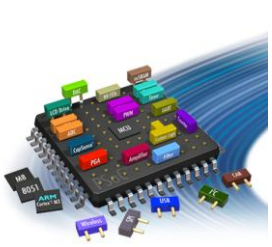
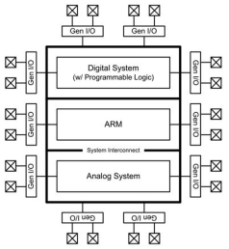
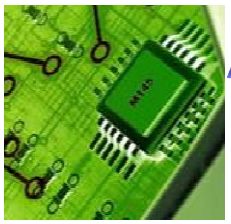
En tête

Le nombre en hexa sur 2 bytes

La chaine contenant le résultat en texte

Marqueur de fin

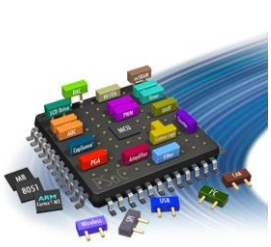
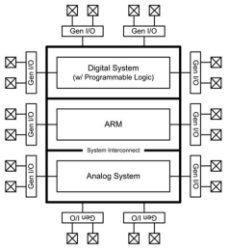
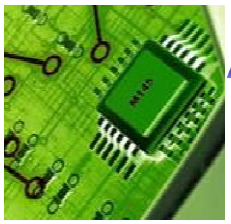
Formation PSoC

```
//*****************************************************************************
//   ANALYSE DE LA TRAME RECUE RS232
//*****************************************************************************
procedure Analyse_Trame_RS232(var ChaineRecue:string);
var
   DoubleDigit : string[2];
   Octet_lo    : byte;
   Octet_hi    : byte;
   i:integer;
   Temp : real;
   Reponse : string;
begin
   Reponse := '';

// Traitement des deux octets en hexa du nombre reçu
   Octet_lo:= byte(ChaineRecue[4]);
   Octet_hi:= byte(ChaineRecue[3]);
   Temp := (Octet_hi*256 + Octet_lo);
   Reponse := floattostr(Temp);

// Renvoi vers les autres unité du programme
   RS232_chaine_recue:=Reponse;


// Forceage de l'écriture du tampon de caractères dans le fichier disque
   writeln(FichTest,'Nombre reçu  : ');
   writeln(FichTest,'hi ',Octet_hi);
   writeln(FichTest,'lo ',Octet_lo);
   writeln(FichTest,'Temp ',Temp);
   writeln(FichTest,'Reponse ',Reponse);
   i:=flush(FichTest);
end;
```

On peut envoyer directement à l'afficheur LCD les caractères reçus, contenus dans une string nommée chaine.

```
// Affichage de la chaine de caractere reçue
// caractères n° 5 à 10 de la trame réceptionnée
   Chaine1:='';
   for i:=5 to 10 do  Chaine1[i-4]:= Rx_RS232[i];
// on met la longueur de la chaine (6 car + long) = 7 car
   Chaine1[0]:= char(7);
// On affiche la chaine dans le champ Edit2
   Edit2.Text := Chaine1;
```


8 Réception de caractères par le PSoC

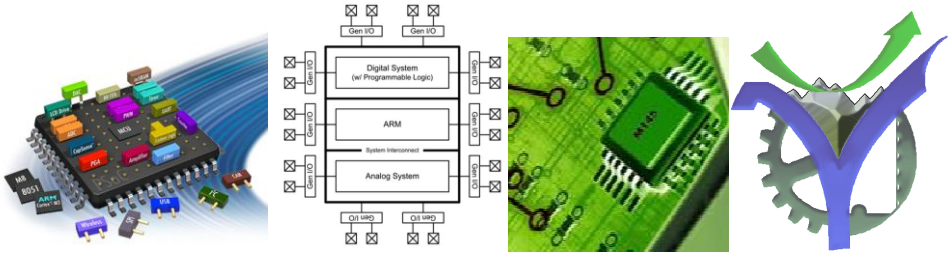
Nous allons maintenant mettre en œuvre l'échange de données dans le sens Delphi => PSoC

Le résultat s'affiche en première ligne de l'afficheur LCD.

Le principe est de détecter l'arrivée d'un caractère, lorsque celui-ci arrive alors on l'accumule dans un tableau nommé tampon. Si le caractère de fin d'envoi 0x0D est détecté alors cela provoque l'affichage des caractères reçus sur la ligne 0 de l'afficheur LCD.

Le pointeur d'accumulation est remis à zéro pour se préparer à une nouvelle acquisition.

Voilà le code PSoC :



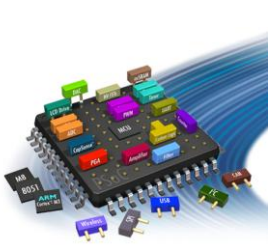
PSoC

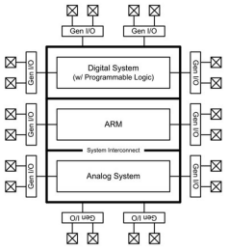
```
// Test de la réception d'un caractère
// Réception
ch = UART_1_GetChar();

// Si il y a un caractère reçu il faut l'accumuler dans un tableau
if( ch > 0) { tampon[count]=ch; count++; }

// Si le caractère reçu est le caractère de fin 0x0D ( retour chariot )
// Alors on affiche le message
if ( ch == 0x0D )
{
    CharLCD_Position(0,0);
    // Ecriture du message
    for (i=0;i<count-1;i++) CharLCD_PutChar(tampon[i]);
    // On efface le reste de la ligne du LCD
    for (i=count+3;i<20;i++) CharLCD_PutChar(' ');
    count=0;
    }
```

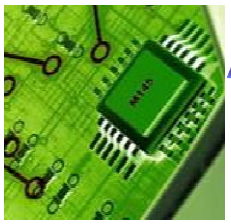
La procédure d'envoi en Delphi est très simple :

Delphi

```
procedure TForm1.Button4Click(Sender: TObject);
begin
// Envoi sur l'UART
   ComPort1.WriteStr('Retour '+ RS232_chaine_recue + chr(13));
```







Indique un document ressource

Retour au sommaire

Retour à la page courante