Tp logiciel C++
Durée: 6 h

TP 4 - Commande d'un afficheur LCD par liaison série asynchrone

Objectifs :

- Gérer un port de communication avec quelques fonctions API.
- Etablir une nouvelle classe LCD_RS232, implémenter les méthodes et procéder aux tests unitaires.
- Analyser la notice technique constructeur du circuit CLCD162 afin de concevoir un interface permettant de commander le module d'affichage LCD.

But :

- Développer un programme destiné à envoyer des messages à un afficheur CLCD162.
- Mise en oeuvre des connaissances théoriques, sur le fonctionnement d'une liaison RS232, et procéder à la configuration de l'UART du PC en écriture à l'aide des fonctions API de communication.
- Définir une nouvelle classe LCD_RS232 qui permet l'utilisation simplifiée de la commande de l'afficheur LCD par liaison asynchrone RS232.

Critères d'évaluation :

Vous serez évalués sur les critères suivants:

* Votre autonomie à résoudre les différents problèmes énoncés.
* Votre capacité à mettre en oeuvre le travail étudié.
* Votre capacité à utiliser le matériel mis à votre disposition.
* La rédaction de l'algorithme et du programme de mise en oeuvre.

Travail demandé

1) En vous aidant de la notice technique constructeur du circuit CLCD162, procéder au câblage de l'alimentation, du circuit d'interface MAX232 et du module LCD puis relier la liaison série RS232 établi au port de communication COM libre d'un PC.

notice constructeur: CLCD162.pdf

2) Tester votre liaison et le module en utilisant le programme "**Putty**".

Le format de la liaison série du module LCD sera :

115200 bauds, 8 bits de données, sans parité, 1 bit de stop

Remarque: ne pas oublier de vérifier la configuration des switchs du module en fonction de la vitesse de transmission choisie.

Programmation de la liaison COM d'un PC sous Qt :

La configuration de l'**UART** ,1'écriture, ou la lecture dans ses registres sont réalisées à l'aide des fonctions **API** de communications. On entend par **API** (**Application Programming Interface**), l'ensemble des fonctions systèmes de l'OS qui peuvent être appelées. Sous Qt, la classe **QextSerialPort** fournit des classes pour le contrôle des ports série.

QextSerialPort fournit une interface de vieux ports série façonné pour les applications basées sur Qt. Il prend actuellement en charge Mac OS X, Windows, Linux, FreeBSD.

Les classes utiles sont:

PortSettings	Contenir les paramètres de port
QextPortInfo	Contenant des informations sur le port
QextSerialEnumerator	Liste des ports disponibles dans le système
QextSerialPort	Encapsule un port série à la fois sur les systèmes Windows et POSIX

QextSerialPort offre à la fois des méthodes interrogation et événementielle pour les API.

Manipulation, sous Qt, de la classe QExtSerialPort:

Pour commencer, il faut installer la librairie **QextSerialPort**. Pour cela, copier dans votre répertoire de projet de développement, le répertoire **\src** qui contient toutes les librairies nécessaires à la commande d'un port série.

Puis, dans le fichier **.pro** de votre projet, on ajouter le fichier **qextserialport.pri** en appelant include

```
...
include(src\qextserialport.pri)
...
```

La manipulation de votre port série s'effectue par l'intermédiaire d'un objet **QextSerialPort**,

- Instancier l'objet
- Configurer la liaison à l'aide des méthodes:

```
void setPortName(const QString &name);
void setQueryMode(QueryMode mode);
void setBaudRate(BaudRateType);
void setDataBits(DataBitsType);
void setParity(ParityType);
void setStopBits(StopBitsType);
void setFlowControl(FlowType);
void setTimeout(long);
```

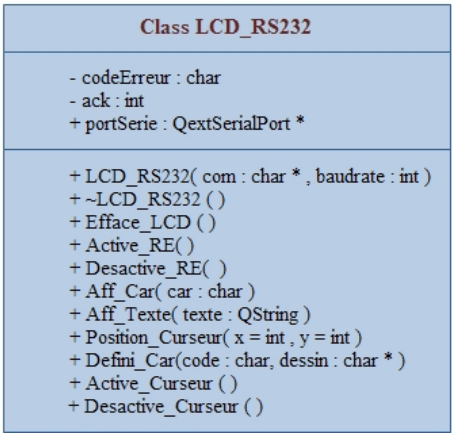
- Communiquer avec votre port série à l'aide des méthodes:

```
qint64 readData(char *data, qint64 maxSize);
qint64 writeData(const char *data, qint64 maxSize);
```

Pour des informations complémentaires, consulter le lien internet :

http://docs.qextserialport.googlecode.com/git/1.2/qextserialport.html

3) Développer une nouvelle classe "**LCD_RS232**", en C++, tel que:

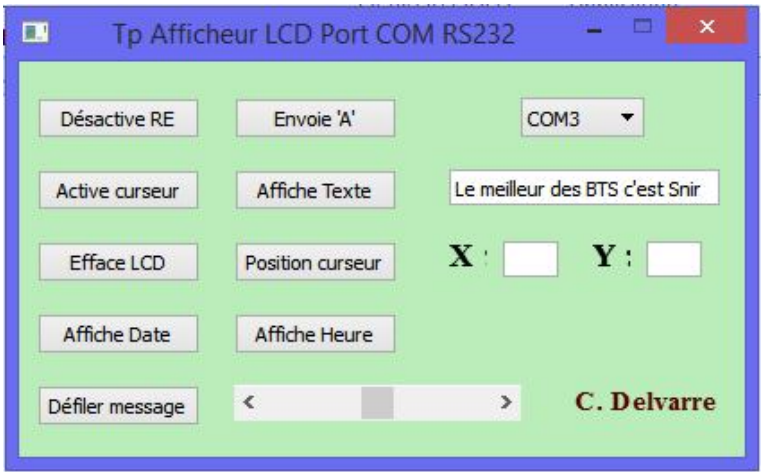


- Le constructeur instancie l'objet portSerie, configure et ouvre le port de communication en écriture (ouvrir une boîte de dialogue pour indiquer le succès de l'opération).
- le destructeur ferme le port de communication utilisé.
- les méthodes, décrites ci-dessus, répondent à la commande de l'afficheur CLCD162.

4) Application pour tester votre classe LCD_RS232

A partir d'une nouvelle application, tester votre classe **LCD_RS232** en instanciant un objet **lcd** et piloter l'afficheur CLCD162 pour obtenir:

- sur la première ligne, la date au format : "jour date mois année",
- sur la seconde ligne, l'heure actualisée dans un format : " hh:mm:ss ",
- sur la quatrième ligne, un message issu d'un objet QLineEdit qui défile à la vitesse définie par un objet TscrollBar.



Bon courage ...

Le Protocole RS232

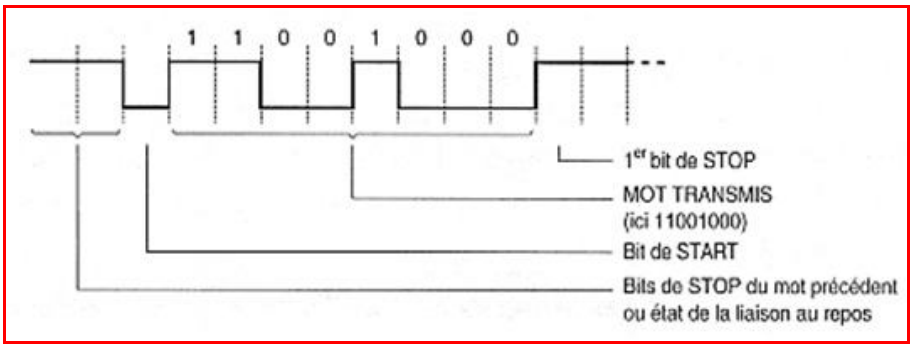
Le **protocole RS232** est apparu en 1962, et bien qu'il est vieux, il est toujours beaucoup utilisé en industrie pour la communication entre un ordinateur via un port série et un système électronique. Les communications sont en Full duplex, c'est-à-dire capable de fonctionner dans les deux sens, ainsi la carte peut envoyer des informations et en recevoir.

La liaison RS232 est un protocole de transfert de données asynchrones qui utilise pour se faire un fil de signal et un fil de masse, la norme RS232 précise les niveaux électriques des signaux chargés de véhiculer l'informations et ajoute à la donnée envoyée un certain nombre de signaux de contrôle.

Pour définir la liaison asynchrone, voyons d'abord le fonctionnement d'une liaison synchrone. La liaison synchrone est un mode de communication à 3 fils, un fil de donnée, un fil d'horloge et un fil de masse. Chaque coup d'horloge indique qu'un nouveau bit vient d'être transmis. Il est donc indispensable dans ce mode de communication que l'horloge soit parfaitement synchrone par rapport à la donnée envoyée.

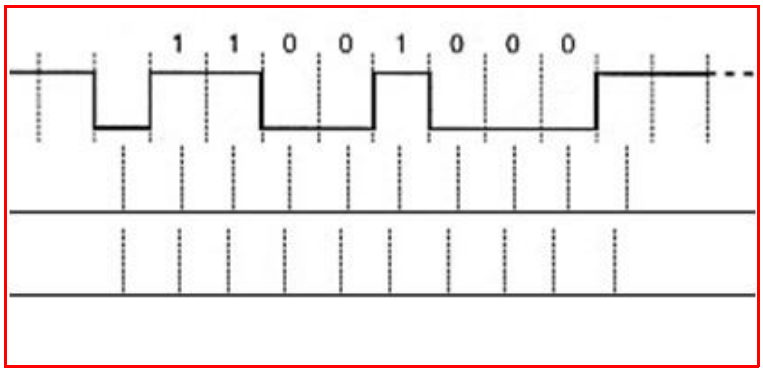
L'idée dans la communication asynchrone est de supprimer cette horloge, afin d'une part de diminuer la taille du bus pour ne passer qu'à un canal indispensable pour transporter l'information (ce qui pourra alors facilité la mise en place d'un système radio) en mettant à l'émetteur et au récepteur une horloge indépendante de chaque coté qui ne sont pas obligées d'être synchrone et dont les fréquences peuvent différer de quelques pourcents sans pour autant empêcher la communication. La liaison asynchrone est donc en théorie beaucoup plus simple à mettre en œuvre.

Pour établir correctement la communication il est nécessaire d'avoir une synchronisation entre l'émetteur et le récepteur. Pour cela il est nécessaire que l'émetteur envoie un signal indiquant au récepteur qu'une trame de donnée va être envoyée. A partir de ce moment tout n'est plus qu'une question de timing. Le protocole et la norme précise la forme du signal transportant l'information ainsi que le débit et la taille de celle-ci. Le débit de l'information est donné en Bauds, 1 baud égale 1 bit par seconde, dans ce cas, le niveau indiquant la valeur du bit envoyer restera présent sur la ligne pendant 1 seconde. Une fois ce temps écoulé, l'émetteur met la ligne au niveau du second bit à envoyé, le récepteur possédant lui aussi une horloge sait quand il doit regarder et enregistrer l'état de la ligne pour en déterminer correctement le bit envoyé et ainsi de suite jusqu'à la fin du mot... Un petit dessin valant mieux qu'un grand discours :



Précisons pour comprendre ce dessin, que premièrement le récepteur et l'émetteur sont configuré de la même façon, ainsi chacun connaît : la vitesse de transmission, l'état de la ligne au repos, la logique correspondant aux niveaux de la ligne, et la taille du mot envoyé (la donnée plus les éventuelle bits de vérification). Dans le cas de ce dessin, il n'y a pas de bits de vérification, la ligne au repos est au niveau haut, l'état du bit 1 est le niveau haut, et la transmission est finie par 1 bit de stop.

Au départ nous avons donc, une ligne à l'état de repos, ou du moins au niveau haut depuis un temps suffisamment long indiquant que toute transmission précédent est terminée. Une nouvelle communication va alors commencer par un basculement de la ligne à l'état opposé à celui du repos (soit l'état bas dans notre cas) pendant une durée de 1/Vitesse en bauds [sec] ce que nous appellerons une période d'horloge. Le récepteur sait alors qu'il va recevoir une information, il démarre alors son horloge, vu que son horloge est à peu de chose près égale à celle de l'émetteur, à chaque coups d'horloge la ligne portera le nouveau bit. Et après 9 coups d'horloge sur le récepteur, celui-ci sait qu'il à reçu le mot et se mettra en attende d'un bit de « start ». Pour se faire correctement il sera préférable que l'horloge du récepteur se décale d'une demi-période par rapport à l'horloge de l'émetteur afin d'aller lire la valeur présente sur la ligne lorsque celle-ci est à la moitié de sa vie et donc assurer que premièrement le signal à eu le temps de s'initialiser correctement et secondement que si les deux fréquences d'horloge diffère légèrement que l'on ne risque pas de lire le mauvais bit.



Les pointillés de la première ligne représente l'horloge de l'émetteur, ceux de la deuxième ligne, l'horloge d'un récepteur possédant une exactement la même fréquence décalée d'une demi période, sur la troisième ligne, la fréquence est différente

Nous remarquons dans le cas ci-dessus ou l'horloge est légèrement plus rapide que celle de l'émetteur, que le récepteur va toujours bien lire à un moment où le signal est bien représentatif du bit envoyé. Dans le cas où nous envoyons dix bits, que le récepteur se synchronise à la demi période, pour que l'horloge du récepteur tombe toujours au bon moment, il ne faut pas qu'elle ne bouge d'un vingtième de période (1/2 * 1/10) par bit, sinon le dernier coup d'horloge risque de tomber mal, soit la fréquence du récepteur doit être la fréquence de l'émetteur à 5% près.

Dans le cas de l'émission la seule chose à faire attention c'est d'avoir la fréquence d'émission correcte, mais nous n'aurons pas à la synchroniser vu que c'est l'émission qui joue le rôle de synchronisation pour le récepteur.

La norme RS232 de L'EIA :

Maintenant que nous avons vu le principe élémentaire d'une liaison asynchrone, voyons la particularité d'une liaison RS232 :

Premièrement elle précise l'état des niveaux étant de -25Và -3V pour caractériser le niveau 1 et 3V à 25volt pour caractérisé le niveau 0.

Deuxièmement, la norme précise le type de connecteur utilisé qui est soit un connecteur de type DB9 soit de type DB25, aux pins de ces connecteurs sont définies les différents signaux de la norme rs232. Nous remarquerons qu'il y en a plus que 3 (deux fils pour le data (émission et réception) et fil de masse) car d'autres signaux ont été ajouté mais ne sont pas obligatoire dans le cadre d'une liaison rs232).

Broche	Signal	Description	E/S
1	CD	Détection de porteuse	Entrée
2	RX	Réception de données	Entrée
3	TX	Emission de données	Sortie
4	DTR	Terminal de données prêt	Sortie
5	GND	Masse de signal	
6	DSR	Données prêtes	Entrée
7	RTS	Requête d'émission	Sortie
8	CTS	Prêt pour l'émission	Entrée
9	RI	Indicateur d'appel	Entrée

Et **troisièmement** la norme définit un protocole :
Afin que les éléments communicants puissent se comprendre, il est nécessaire d'établir un protocole de transmission. Ce protocole devra être le même pour les deux éléments afin que la transmission fonctionne correctement.

Paramètres rentrant en jeu :

- **Longueur des mots** : 7 bits (ex : caractère ascii) ou 8 bits

- **La vitesse de transmission** : les différentes vitesses de transmission sont réglables à partir de 110 bauds (bits par seconde) de la façon suivante : 110 bds, 150 bds, 300 bds, 600 bds, 1200 bds, 2400 bds, 4800 bds, 9600 bds,...

- **Parité** : le mot transmis peut être suivi ou non d'un bit de parité qui sert à détecter les erreurs éventuelles de transmission. Il existe deux types de parité:

parité paire : le bit ajouté à la donnée est positionné de telle façon que le nombre des états 1 soit paire sur l'ensemble donné + bit de parité.
ex : soit la donnée 11001011 contenant 5 état 1, le bit de parité paire est positionné à 1, ramenant ainsi le nombre de 1 à 6.
***parité impaire* :** le bit ajouté à la donnée est positionné de telle façon que le nombre des états 1 soit impaire sur l'ensemble donné + bit de parité.
ex : soit la donnée 11001001 contenant 5 état 1, le bit de parité paire est positionné à 0, laissant ainsi un nombre de 1 impaire.

- **Bit de start** : la ligne au repos est à l'état logique 1 pour indiquer qu'un mot va être transmis la ligne passe à l'état bas avant de commencer le transfert. Ce bit permet de synchroniser l'horloge du récepteur.

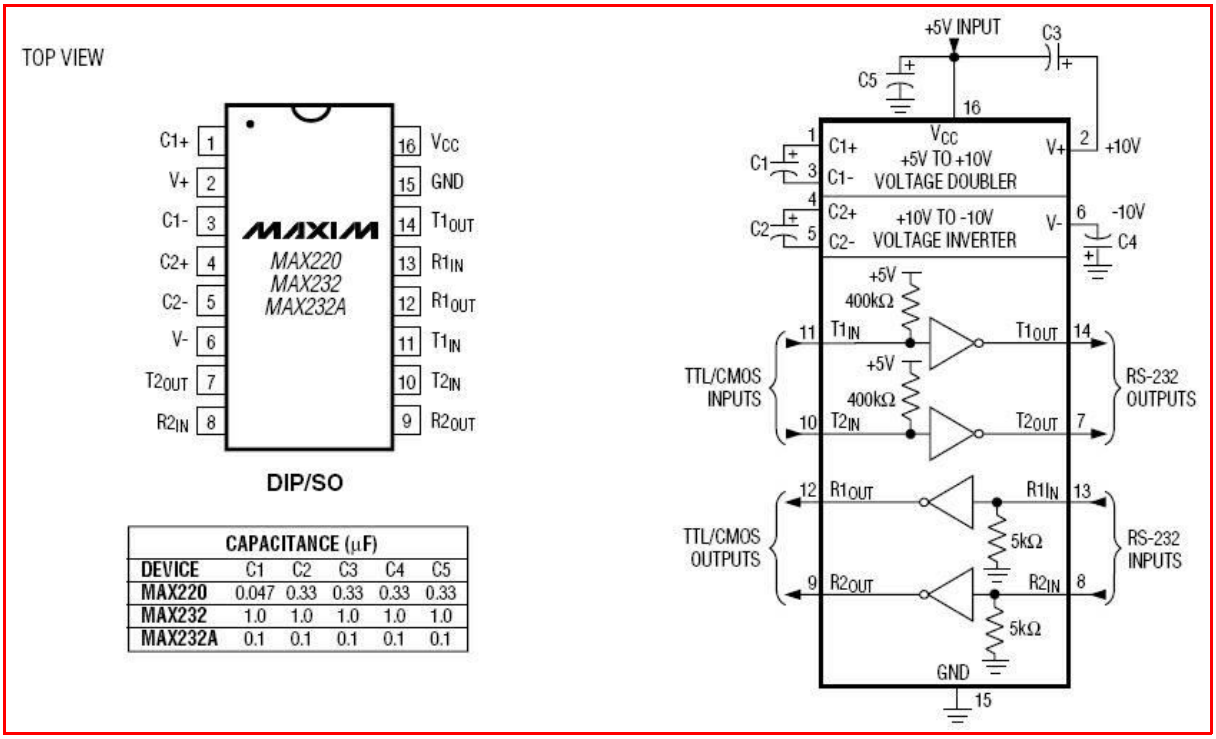
- **Bit de stop** : après la transmission, la ligne est positionnée au repos pendant 1, 2 ou 1,5 périodes d'horloge selon le nombre de bits de stop.

Ensuite nous finirons par prévenir que la liaison RS232 sur le câblage prévu à cet effet est censé porter sur 10m, et est capable de monté dans certaines conditions dans lesquelles nous ne rentrerons pas sur 100m. Ce qui rend la norme RS232 énormément utilisée comme bus de terrain pour communiquer entre un ordinateur et un drive ou autre chose dans le cadre par exemple du contrôle d'une machine. Les nouvelles normes telle que l'usb ont des portées nettement moins importante mais avec l'apparition des liaisons sans fils la liaison rs232 tente à laisser le relais. Pour ce faire, il est apparu sur le marché au cours des dernières années des convertisseurs USB vers série, telle que le FTDI ou encore le 18f4550 qui permetten de simuler un port série sur un pc (au travers d'un driver). L'interface alors se connecte sur le port USB mais est reconnue comme un port série sur l'ordinateur, ce qui permet alors d'utiliser les anciens logiciels de contrôle avec une interface électronique plus récente.

La conversion des niveaux :

Les cartes électroniques à base de microcontrôleurs fonctionnent très souvent avec des niveaux TTL soit 0-5Volt, 0V pour le niveau 0 et 5Volt pour le niveau 1. Brancher donc directement une ligne RS232 sur un microcontrôleur n'aurait donc aucun sens et pourrait aussi endommager le système en imposant des tensions de 25volt.

Pour rendre compatible une ligne RS232 avec une carte de ce type il existe un composant très simple d'utilisation que nous allons étudier : le max232. En regardant son schéma interne ci-dessous, nous constatons directement qu'il est premièrement doté d'un convertisseur de tension, au travers des capacités C1 et C3 il génère une tension de 10Volt depuis les 5Volt (doubleur de tension), et au moyen des capacités C2 et C4 il génère une tension de -10Volt a partir de la tension de 10Volt. Il est bien sur évident que la puce est munie de tout un système, avec un oscillateur, des diodes et ... afin d'intégré ce convertisseur DC-DC. Il existe une version de cette puce, le max233, où les capacités sont intégrées directement dedans, mais nous ne rentrerons pas dans ce détail. La valeur des capacités va dépendre de la version de la puce :



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

La connexion avec le port série se fait via la broche Rx (réception) Tx (émission) et RTS qui donne la possibilité d'émission. Les broches 6, 1 et 4 seront reliées ensembles, Et nous n'oublierons pas de brancher la masse (broche 5) avec la masse du circuit. Les broches 7 et 9 ne sont pas nécessaires dans notre cas.