Transmission de données en série et mise en œuvre sur le 9s12

Module Info 2

Sebastien.Kramm@univ-rouen.fr

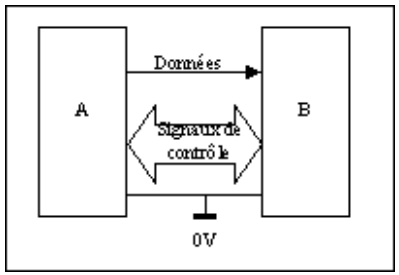
IUT GEII Rouen

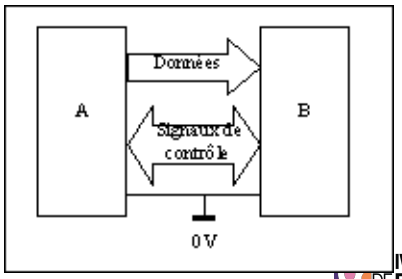
2013-2014

Sommaire

Transmission d'information entre 2 systèmes informatisés

- Distance : 2 cm, 2 m, 2 km..., voire des Mm ou Gm !
- 2 approches :
  - Parallèle : 'n' fils (8, 16, 32, ...) + signaux de contrôle.
    ⇒ Distances **courtes** (liaison interne au système, ou périphérique proche) et/ou beaucoup de données (video).
  - Série : données transmises sur 1 seul **canal de transmission** (fil électrique, fibre, liaison radio, etc.)
    ⇒ Toutes distances.





Exemples de liaisons parallèles

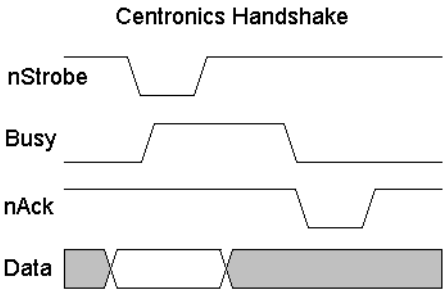
- Transmission "intra système" : bus IDE/PATA, PCI, AGP (*Accelerated Graphics Port*).

- Transmission vers périphériques
  - "Port parallèle" des PC (connecteur DB25) : permet les connections en "IEEE 1284" :
    - SPP (*Standard Parallel Port*)
    - EPP (*Enhanced Parallel Port*)
    - ECP (*Extended Capability Port*)
  - IEEE 488 (bus de mesure)

⇒ en voie d'obsolescence...

Handshake sur liaison parallèle

- Il faut un protocole de synchronisation entre émetteur et récepteur.
- Exemple : protocole "Centronics" (1970 !) :
- Pour chaque octet :
  1. L'émetteur place les données sur le bus, et active le signal **strobe**
  2. Le récepteur active **busy** ("occupé"), et traite (lit) les données
  3. Quand il a fini, il désactive **busy**, et génère une impulsion sur **ack** (*acknowledge* = acquittement)



Exemple de liaisons séries

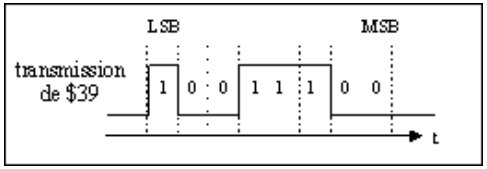
- Applicatifs beaucoup plus vaste
  - Supports non-électriques : GSM, satellite télécom, télévision (TNT), etc.
  - Support électrique
- Normes "couches basses"
  - Asynchrone : RS232, RS422, RS485
  - Synchrone : I2C, SPI
- Protocoles évolués : SATA, Ethernet, USB, xDSL,...
- Les liaisons séries surpassent aujourd'hui les parallèles !

- Interface HD "P-ATA"

- Interface HD "S-ATA"

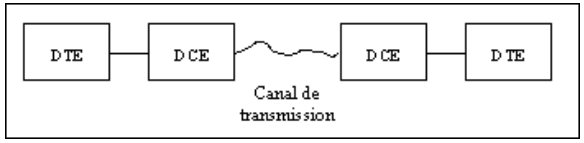
Transmission de données en série : principes

- Principe : les bits sont transmis un par un sur le canal de transmission (**LSB en premier**).



- Pour les grandes distances, on utilise les termes :
  - DTE : *Data Terminal Equipment* ⇒ Ordinateur
  - DCE : *Data Communication Equipement*
    - Adaptation signal au canal (codage & modulation)
    - Gestion de la liaison (établissement, maintien, libération)

  ⇒ "Modem", "Box", ...

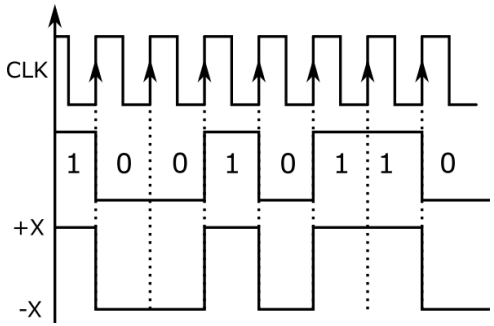


Adaptation au canal de transmission

- La transmission sur le canal physique peut se faire :
  - directement, ou après un **encodage** des niveaux
    ⇒ transmission en **bande de base**,
  - via une **modulation** : le signal à transmettre va modifier une **porteuse**, signal de fréquence beaucoup plus élevée.
    (obligatoire pour les canaux non-électriques : radio, fibre, ...).
- Ces tâches sont dédiées au **DCE**, aussi appelé **modem** (modulateur/démodulateur).

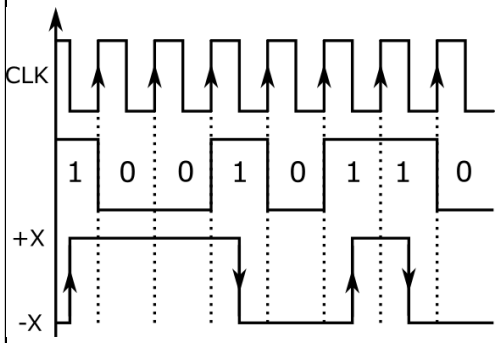
Codage des niveaux

- Pour adapter le signal au canal de transmission, on **code** les niveaux logiques.
- Objectifs :
  - minimiser la bande passante occupée, et la décaler vers le haut du spectre,
  - permettre la récupération de l'horloge (transmissions synchrones),
  - détecter la présence ou l'absence du signal.
- Codage le plus simple : NRZ (No Return to Zero).



- '0' logique : +X
- '1' logique : -X
- Permet la détection d'absence de signal

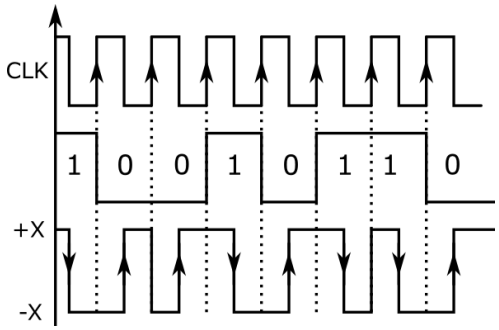
Codage NRZI



- 0 : pas de changement du signal
- 1 : front entre 2 tops horloge

- Utilisé dans la norme USB.
- Inconvénient : risque de perte de synchronisation si longue transmission de '0'.
  ⇒ Solution : au bout de 6 bits à 0, on ajoute un bit à 1 (technique du *stuffing*).

Codage Manchester

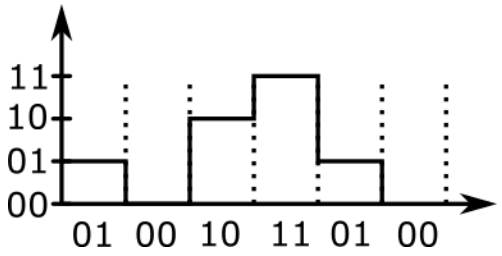


- Chaque bit est codé par un front au milieu de la période :
- 0 : front montant
- 1 : front descendant

- Intérêt : synchronisation de l'horloge du récepteur sur l'émetteur facilitée.
- Problème : si inversion des fils, inversion des bits ...
  ⇒ création du "**Manchester différentiel**".

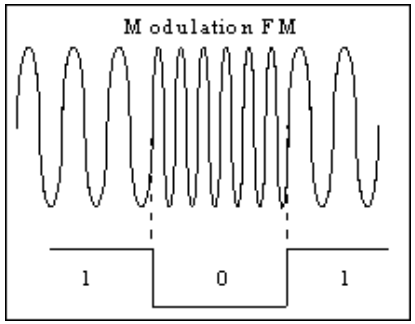
Codage multi-niveaux

- Si on peut distinguer 4 niveaux différents sur le support, on peut transmettre 2 bits par période d'horloge.
- On peut ainsi **doubler le débit** (exprimé en bits/s.), sans augmenter la cadence d'horloge (exprimée en Bauds).



- ⇒ Transmission de 12 bits en 6 périodes d'horloge.

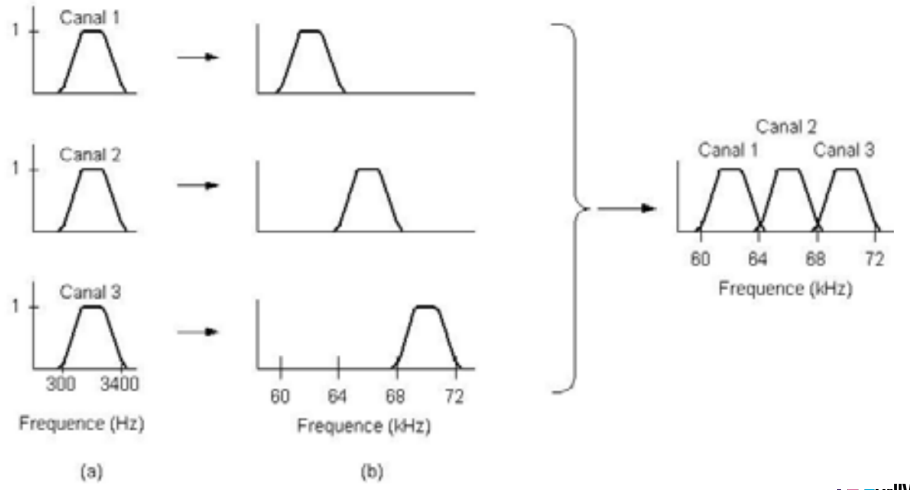
Modulation du signal



- Types de modulation :
  - modulation d'amplitude (AM), ou OOK (AM en "tout ou rien"),
  - modulation de fréquence (FM), en numérique : FSK (*Frequency Shift Keying*),
  - modulation de phase.
- En pratique, on utilise des modulations plus complexes, qui combinent ces techniques pour transmettre plusieurs bits par période.
  - ASK : *Amplitude Shift Keying*, modulation par saut d'amplitude,
  - PSK : *Phase Shift Keying*,
  - QAM : *Quadratic Amplitude Modulation*,
  - ...
- Interêt : le signal occupe une largeur de spectre limitée.
  $\Rightarrow$ On pourra **multiplexer** différents signaux sur le même canal.

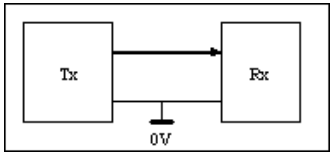
Multiplexage fréquentiel

- On translate chaque signal dans le domaine fréquentiel, avant de les mélanger.

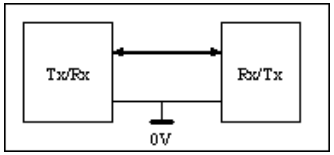


Modes de transmission

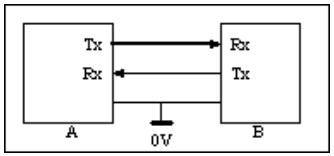
- Liaison "**Simplex**" :
  circulation des données dans un seul sens.



- Liaison "**Half Duplex**" :
  bidirectionnelle, mais un seul canal de transmission, utilisé alternativement dans chaque sens.

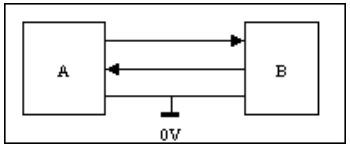


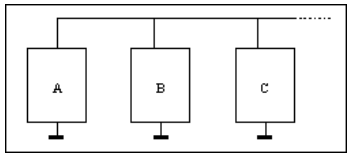
- Liaison "**Full Duplex**" :
  bidirectionnelle & simultanée (2 canaux de transmission)
  Remarque : possible dans certains cas avec un seul canal



Topologie de connexion : deux cas de figure

1. Liaison "point à point"
2. Plusieurs sous-ensembles (μP, périphériques, ...) interconnectés sur le même **support** (fil, bande de fréquences radio, etc.)
   $\Rightarrow$ On parle alors de **bus**, il faut des mécanismes
   - de gestion d'adresse,
   - de gestion des conflits d'accès au support.





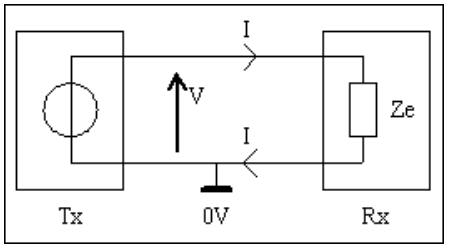
Support électrique : 2 méthodes

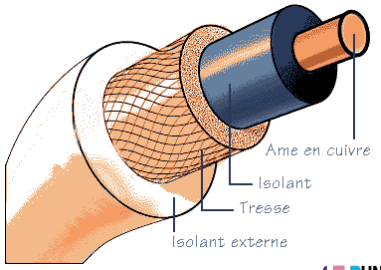
- La transmission du signal peut se faire de deux façons :
  - En "**mode commun**" : 2 fils, signal + masse,
  - En "**mode différentiel**" : 3 fils, signal sur 2 fils en opposition de phase + masse.
- Les performances (débit, longueur maxi) sont liées aux caractéristiques du câble.
- Temps propagation typique : $5\mu s/km$

Transmission en mode commun

- Principe : 2 fils : signal + masse.
- Mauvaises performances en vitesse si impédance pas adaptée (réflexions en bout de ligne).
- Sensibilité au perturbations électromagnétiques si câble pas blindé.
- Diaphonie importante si plusieurs canaux sont proches.

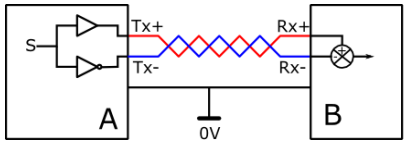
⇒ Limité au faibles distances (quelques mètres).

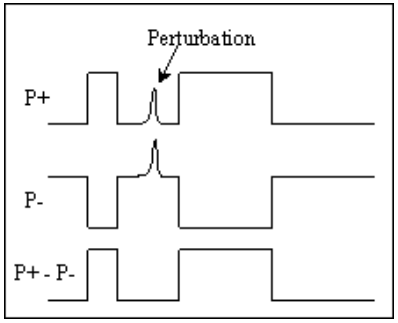




Transmission en mode différentiel

- Le signal utile est transmis sur 2 fils en opposition de phase.
- La référence de tension (masse 0V) est optionnelle.
- Meilleures performances (débit, distance).
- Une perturbation influant sur les 2 fils sera "supprimée" (en réalité, atténuée) par soustraction.

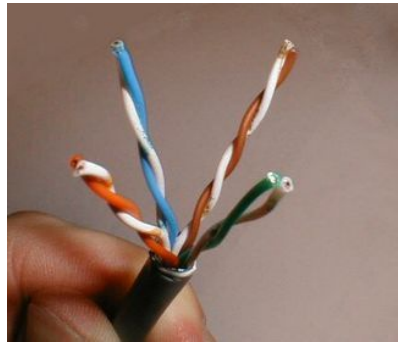




Mode différentiel : câblage

- Ce type de transmission se fait en utilisant de la **paire torsadée** : perturbations électromagnétiques réparties sur les 2 fils.
- En général, les câbles regroupent plusieurs paires, plus une masse commune.

- Plusieurs types de câbles :
  - *Unshielded Twisted Pair* (**UTP**),
  - *Shielded Twisted Pair* (**STP**) pour la version blindée par paire,
  - *Foiled Twisted Pair* (**FTP**) pour la version avec un blindage global.
- Les câbles sont classés en catégories, suivant leur performances (Cat. 4, 5, 6...)
- Utilisé pour les réseaux téléphoniques et informatiques.



Aspect temporel : deux approches possibles

Liaisons séries **asynchrones**

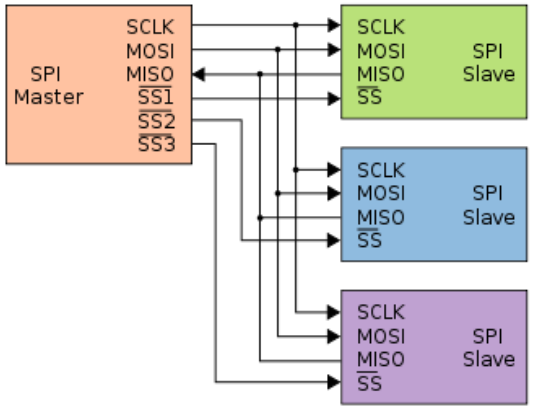
- Pas de synchronisation, mais Tx et Rx doivent avoir une horloge de fréquence identique.
- Le début de chaque émission est précédé d'un **signal** (bit ou octet) permettant au récepteur de "se caler" sur l'émetteur.

Liaisons séries **synchrones**

- L'horloge est transmise en même temps que le signal.
- Meilleures performances (débit, distance maxi).

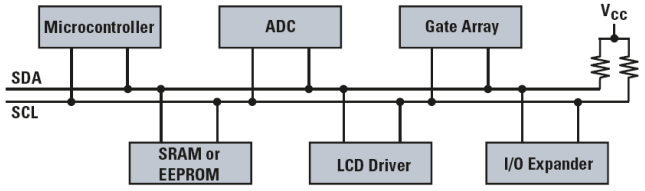
Liaisons séries synchrones

- 3 fils (signal + horloge + masse), mais parfois 2 fils (horloge mélangée au signal)
- Principe : données synchronisées sur une horloge
- Exemple de norme : SPI, I2C, ...
- Permettent parfois la topologie "bus"



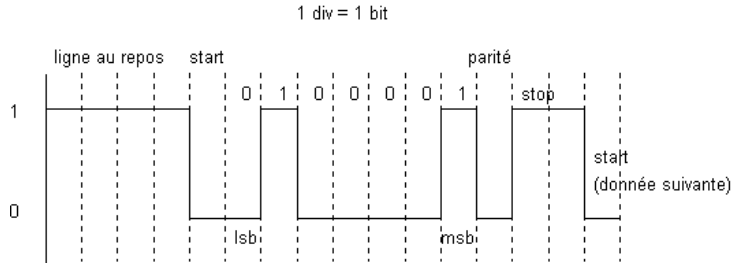
Bus synchrone I2C

- Origine : Philips, fin années 1980.
- Objectif : communication entre CI sur une même carte.
- 2 signaux (+ masse)
  - *Serial Data Line* (SDL)
  - *Serial Clock* (SCL)
- Débits normalisés
  - *Standard mode* : 100 kb/s
  - *Fast mode* : 400 kb/s
  - *High Speed mode* : 3,4 Mb/s



Liaisons séries asynchrones

- Transmission sur 1 canal par sens.
- Parfois, signaux de contrôle supplémentaires (RS232).
- Principe : données transmises de façon asynchrone.
  - ligne au repos à '1',
  - bit de "start" ('0') pour repérer le début de l'octet.
- Tx et Rx calés sur la même vitesse.
- Exemple de normes : RS232, RS422, RS485.



Asynchrone : paramètres importants

- Vitesse de transmission : exprimée en **Bauds** (= bits/s. si on transmet un bit par période).
- Débits normalisés, par ex. : 4800, 9600, 14400, 19200, 28800, 56000, 115200 bauds.
- Nombre de bits : 7 (code ASCII originel), 8, ou 9.
- Bit de stop (optionnel) : signale la fin du mot.
- Bit de parité (optionnel) : permet la détection des erreurs.
- Niveaux électriques.

Niveaux RS232 : inversion

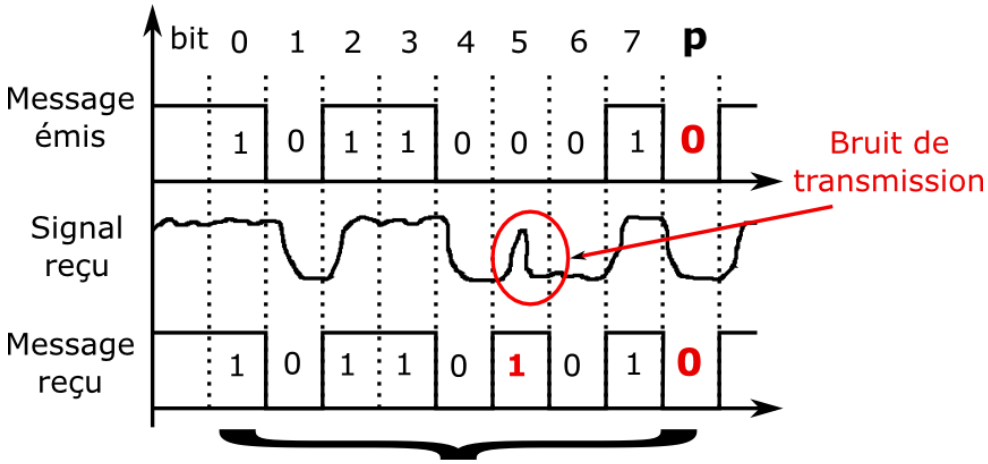
- '0' logique ⇒ + 3 à + 25 V (niveau haut)
- '1' logique ⇒ - 3 à - 25V V (niveau bas)

Bit de parité

- Intérêt : permettre la **détection** des erreurs de transmission, par l'**ajout** d'un bit supplémentaire.
- Principe :
  - le bit est positionné par l'émetteur en fonction du nombre de bits à '1' du message transmis ;
  - à la réception, on vérifie la règle ci-dessous.
- Règle (en mode "parité paire") : **le nombre de bits à '1' du message total (message + bit de parité) doit être pair**.
- Exemples (en mode "parité paire") :

octet à transmettre	bit de parité
0000 0000	
0100 0000	
0110 0000	
1101 0001	

Bit de parité : exemple

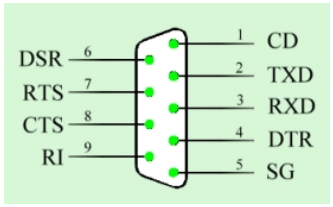


Généralisation de la détection d'erreur

- L'intérêt du bit de parité est limité : il ne permet de détecter une erreur que sur **un seul** bit.
- Des techniques plus performantes existent, toujours en **ajoutant** de l'information au message à transmettre.
  - **somme de contrôle** (*checksum*) : consiste à calculer une somme, qu'on tronque et qu'on ajoute en fin de message.
  - CRC : **Contrôle de Redondance Cyclique** (*Cyclic Redundancy Check*) : on ajoute au message le reste de la division par 2 du message.
- Ces techniques s'appliquent sur un ensemble d'octet et non pas au niveau de chaque octet.

Norme RS232 (EIA 232)

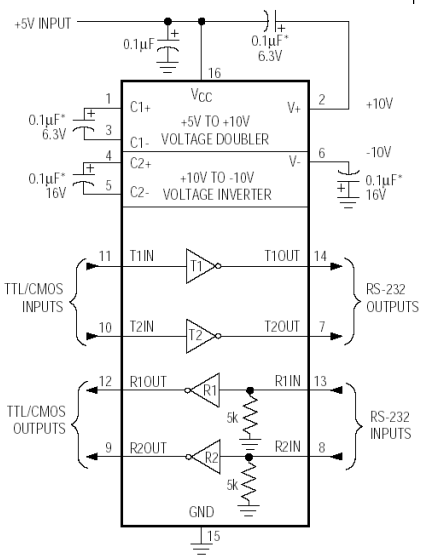
- Sur PC : version "allégée" (ports série COM1 & COM2 des PC, connecteur DB9). ⇒ en voie de disparition...
- Définit uniquement les aspects *hardware* (connectique, électrique), pas les protocoles logiciels.
  - Niveaux référencés par rapport à la masse ("mode commun").
  - Conçu pour liaison "point à point", en *Full Duplex*.
  - Distances : jusqu'à 50 m.
  - Connectique : DB9 ou DB25.
  - Spécification des signaux de contrôle, facultatifs (gestion de la ligne).



- TxD : Emission
- RxD : Reception
- SG : Signal Ground
- Autres signaux : gestion de la communication

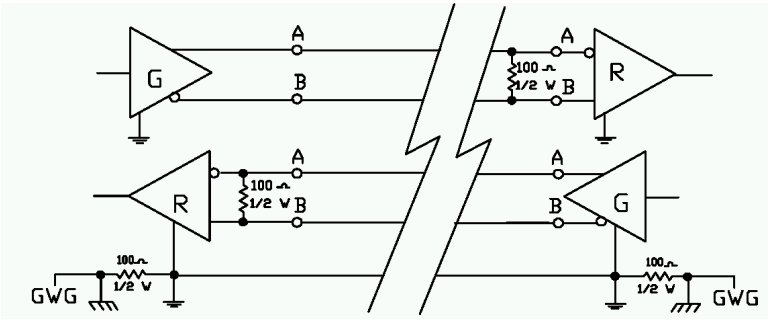
Adaptation des niveaux : CI "MAX 232"

- Permet de réaliser facilement l'adaptation de niveaux.
- Génère en interne ("pompe de charge") les tensions nécessaires (+10 et -10V).
- Economique.
- Devenu un standard de l'industrie.
- Original : Maxim, mais existe chez de nombreux fabricants.



Norme RS422

- Evolution de la RS232 en version différentielle.
- Performances : jusqu'à 1km, 10 Mbits/s.
- Nécessite une résistance terminale de 100 Ω pour adapter la ligne (moins de réflexions en bout de ligne).



Norme RS485

- Adaptation de la RS422 à une topologie "bus".
- Les drivers ont des sorties "3 états" : '0', '1', Hi-Z.

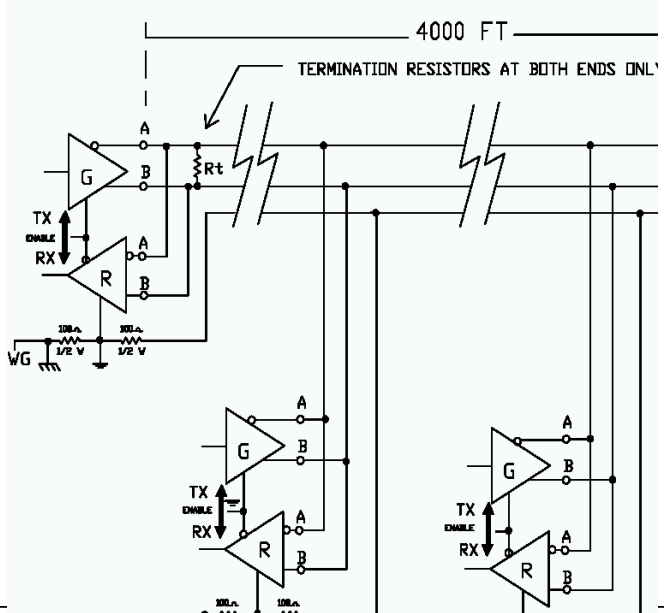


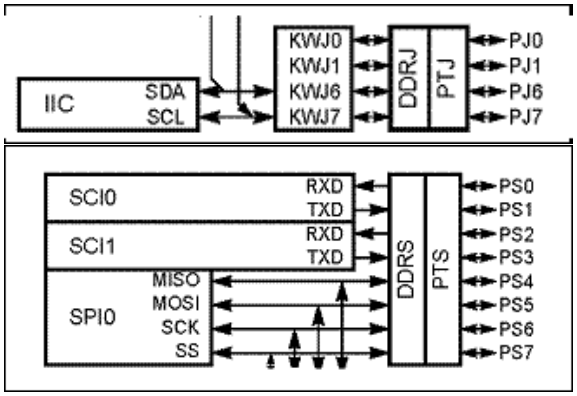
Tableau comparatif

Nom	Debit (théorique)	Distance max.
RS232	20 kb/s	15m
RS485	35 Mb/s 100 kb/s	15m 1200 m
I2C	100 kb/s - 3,4Mb/s	2-3m
USB-2 USB-3	480 Mb/s 5 Gb/s	5m 3m
IEEE1394 (FireWire)	400-800-1600 Mb/s	5m
SATA III	6Gb/s	faible

Sommaire

Liaisons séries sur le 9s12

- Liaisons synchrones : bloc **IIC** (I2C) et bloc **SPI** (*Serial Peripheral Interface*).
- Liaisons asynchrones : 2 blocs **SCI** (*Serial Communication Interface*).



Présentation du SCI (asynchrone)

- Intègre une UART (*Universal Asynchronous Receiver Transmitter*) Full-Duplex.
- Ne gère pas les signaux de contrôle (seulement 2 broches, TxD & RxD).
- Evite d'avoir à programmer les aspects techniques (sérialisation, ...)
- Permet la communication simultanée (Emission - réception).
- Utilisation pratique :
  - Initialisation (configuration des paramètres).
  - Emission d'un octet : appel d'une fonction en lui passant l'octet à émettre.
  - Réception d'un octet : de préférence en interruption.
    $\Rightarrow$ nécessite une fonction de traitement de l'octet reçu.

Registres du SCI : 2 x 8 registres

Name		Bit 7	6	5	4	3	2	1	Bit 0
SCIBDH	R	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
	W								
SCIBDL	R	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
	W								
SCICR1	R	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
	W								
SCICR2	R	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
	W								
SCISR1	R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
	W								
SCISR2	R	0	0	0	0	0	BRK13	TXDIR	RAF
	W								
SCIDRH	R	R8	T8	0	0	0	0	0	0
	W								
SCIDRL	R	R7	R6	R5	R4	R3	R2	R1	R0
	W	T7	T6	T5	T4	T3	T2	T1	T0

= Unimplemented or Reserved

Figure 1-2. SCI Register Summary

- A l'utilisation, on ajoutera '0' ou '1' pour indiquer lequel (exemple : `SCI1CR1 = a;`)

Utilisation pratique du SCI : 6 registres à considérer

4 registres pour la **configuration** de la liaison

- **SCIBDH**, **SCIBDL** : *SCI BauD register High & Low* : sélection vitesse de transmission.
- **SCICR1** : SCI **C**ontrol **R**egister **1** ($00 en général)
- **SCICR2** : SCI **C**ontrol **R**egister **2**

⇒ à programmer **1 seule fois** dans le bloc d'initialisations

2 registres pour l'**utilisation** de la liaison

- **SCIDRL** : SCI **D**ata **R**egister **L**ow (émission & réception)
  ⇒ registre de donnée.
- **SCISR1** : SCI **S**tatus **R**egister 1
  ⇒ registre d'état.

Registre SCISR1 : 8 bits d'états

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
Reset	0	0	0	0	0	0	0	0

- Les 2 bits les plus importants :
  - TDRE : signale que le registre de transmission est vide
    ⇒ On peut écrire l'octet suivant.
  - RDRF : signale que le registre de réception est plein
    ⇒ On peut (on doit...) venir lire l'octet reçu.
- RAZ : procédure en 2 temps
  - TDRE : **lecture** de SCISR1 suivi d'une **écriture** dans SCIDRL.
  - RDRF : **lecture** de SCISR1 suivi d'une **lecture** dans SCIDRL.

Initialisations : 3 choses à faire

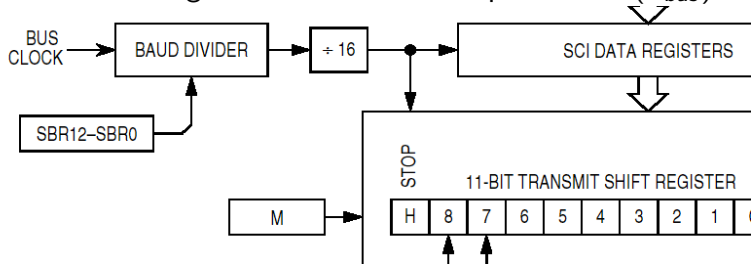
1. Choix vitesse de transmission (registre SCIBDL et H)
2. Indiquer si on veut (registre SCICR2) :
   - Émission seule ⇒ TE = 1, RE = 0
   - Réception seule ⇒ TE = 0, RE = 1
   - Les deux ⇒ TE = 1, RE = 1
3. Validation locale interruptions (registre SCICR2)
   - Sur réception octet (quand RDRF = 1) : RIE = 1
   - Sur fin d'émission octet (quand TDRE = 1) : TIE = 1

- Registre SCICR2 (*Control Register 2*)

	7	6	5	4	3	2	1	0
Read Write	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET:	0	0	0	0	0	0	0	0

Sélection débit de transmission

- L'horloge de l'UART du SCI est générée par division de fréquence à partir de l'horloge de cadencement du processeur (F_{bus})



  ⇒ Pour une fréquence donnée, toutes les débits ne seront pas disponibles !
- Il faut choisir :
  - un débit compatible avec F_{bus} (24MHz pour nous),
  - un débit normalisé.

Débit : registres SCIDBL & SCIBDH

- Le débit réel est donnée par la formule :
  $D = \frac{F_{bus}}{16\ SBR}$
  SBR : valeur (entière) 13 bits [SBR12 :SBR0], dans le registre SCIBDL et SCIBDH
- Exemple : débit "cible" D = 9600 Bauds, avec $F_{bus} = 24$ MHz
  $\frac{24M}{16.9600} =$ _____ ⇒ SBR = _____
  ⇒ débit réel $D_r = \frac{24M}{16*SBR} =$ _____ Bds
  ⇒ erreur $= \frac{|D-D_r|}{D} =$ _____ %
- On pourra tolérer une erreur de quelques %

Utilisation en réception

- Le SCI gère la réception d'**un seul octet** (pas de mémoire tampon)
  ⇒ Dès réception, on doit venir le lire avant que le suivant n'arrive ! (sinon, perte de l'octet)
- Le flag RDRF (*Receive Data Register Full flag*) signale la réception complète d'un octet (registre SCISR1).
- Détection : scrutation ou interruption.

Réception d'un octet en scrutation

- Fonction de lecture d'un octet (sur le SCI n°0) :

```
1  char lire_octet()
2  {
3     while( (SCI0SR1 & BIT5) == 0 ) // attente du passage à 1
4        ;                           //     du bit RDRF
5     return SCI0DRL;                // lecture de l octet reçu
6  }
7
```

Pertinence de la scrutation ?

- Problème : et si l'émetteur n'envoie plus rien ? On reste bloqué...
- De plus, le processeur passe son temps à attendre un octet : pas très efficace !
  ⇒ La réception se fera donc toujours en interruption.
  (validation des interruptions via le bit RIE de SCICR2)

Réception d'octet en interruption

- La routine devra :
  - faire la RAZ du flag,
  - lire l'octet reçu et le traiter.
- Exemple de routine (SCI 0) :

```
1 void isr_sci_0()
2 {
3     char a = SCI0SR1; // pour la RAZ
4     traitement( SCI0DRL ); // fonction de traitement
5 }
```


Utilisation en émission

- L'émission démarre automatiquement, dès l'écriture d'un octet dans le registre de données (SCIDRL).
- Avant, il faut être sûr que le registre soit vide
  (sinon, risque d'écrasement de la donnée précédente).
  ⇒ vérifier que TDRE = 1
  (*Transmit Data Register Empty flag*), registre SCISR1

Emission : scrutation ou interruption ?

- Pour l'émission, la scrutation moins critique :
  le temps passé à attendre est **défini à l'avance** (lié à la vitesse de transmission).
- Exemple : vitesse = 9600 Bauds
  ⇒ La durée de l'émission d'un octet est de
  (8 bits+1 bit de start) / 9600 ≈ 1 ms
- Pas de risque de blocage.
- Fonctionnement en interruption toujours possible si besoin.

Envoi d'un octet en scrutation

- Il faut, à chaque fois :
  - Vérifier (=attendre) que le registre de données soit vide (flag TDRE)
  - Ecrire l'octet dans le registre de donnée (SCIDRL)
- Fonction d'émission d'un octet (SCI n° 0) :

```
1  void envoi_octet( char octet )
2  {
3     while( (SCI0SR1 & BIT7 ) == 0 ) // attente TDRE=1
4        ;
5     SCI0DRL = octet; // emission octet
6  }
```


Sommaire

Exercices : page 1

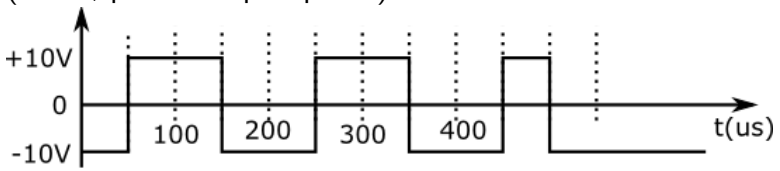
- On observe à l'oscilloscope le signal suivant sur une liaison asynchrone à 8 bits, sans parité ni bit de stop, à un débit de 14400 Bauds.
  - Quel est l'octet transmis ?
  - Donner la période (durée d'un bit)
  - Combien de temps faut-il pour transmettre 1 octet ?

Exercices : page 2

- On reçoit le signal suivant d'une liaison asynchrone à 8 bits, sans bit de stop, mais avec parité paire, avec un débit de 56000 Bds.
  - Quel est l'octet transmis ?
  - Le bit de parité est-il correct ?
  - Donner la période (durée d'un bit)
  - Combien de temps faut-il pour transmettre 1 octet ?

Exercices : page 3

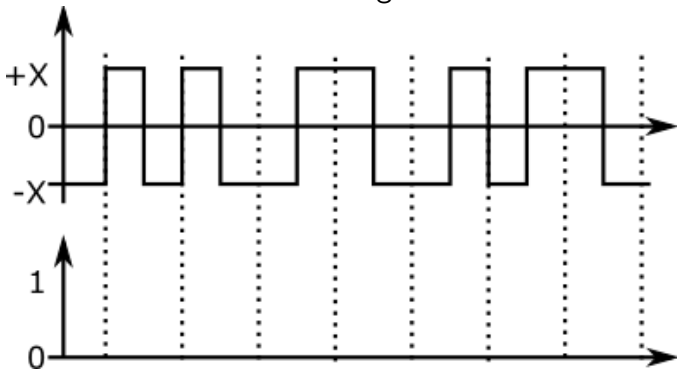
- Sur une liaison RS232, on observe à l'oscilloscope le signal suivant (8 bits, pas de stop ni parité)



  - Quel est l'octet transmis :
  - Quel est le débit en Bauds de la liaison :
  - Quel est le débit normalisé qui se rapproche le plus :
  - Que vaut l'erreur (%) :

Exercices : page 4

- On observe à l'oscilloscope le signal suivant sur une liaison à codage "Manchester". Dessiner le signal binaire encodé.

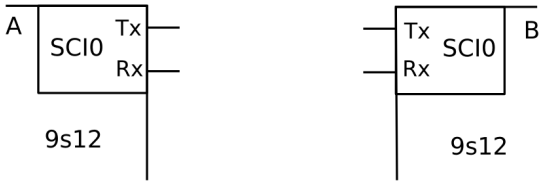


Exercices : page 5

1. Ecrire un prog qui va envoyer sur la liaison série (SCI0) le bloc de 100 octets situé à l'adresse TABLE quand on va appuyer sur un BP (actif à 0) connecté sur PT0.
   (transmission à 56000 Bauds, 8 bits, sans stop ni parité).
2. Soit une liaison numérique via un canal supportant 16 niveaux binaires différents à une cadence de 50000 Bds. Calculer le débit utile, exprimé en ko/s ;
   2.1 Si on transmet en asynchrone avec 1 bit de parité ?
   2.2 Si on transmet en synchrone ?

Exercices : page 6

- On souhaite connecter ensemble deux systèmes par une liaison électrique directe, de façon à permettre la transmission en full-duplex. Dessiner la connexion :



- On souhaite réaliser une transmission respectant la norme RS232. Que faut-il ajouter comme composant ? Dessiner le schéma d'un des cotés. Faut-il un câble droit ou croisé ?