# La liaison UART

- UART =
  - Universal (indépendant de la machine)
  - Asynchronous (aucun composant n’envoie d’horloge)
  - Receive (une ligne dédiée à la réception)
  - Transmit (une (autre) ligne dédiée à la transmission)

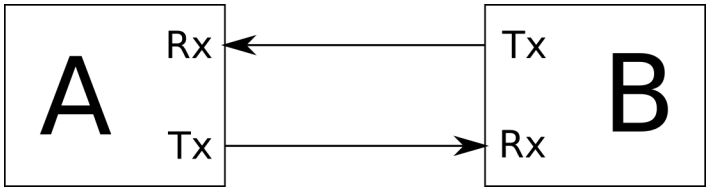
- Commercialisation : début 70’s

# La liaison UART

- Très vite utilisé pour piloter des imprimantes en conjonction avec le codage ASCii
- Utilisé pour les terminaux des serveurs
  - On reste dans le domaine de l’échange de texte
- Extension aux terminaux virtuels
  - Encore d’actualité dans les basses couches linux
- Application aux systèmes embarqués
  - Permet un terminal quand le réseau n’est pas encore fonctionnel
  - Interface par défaut d’un système embarqué (pour le développeur)
  - Toujours active sur de nombreux terminaux (simplement pas accessible physiquement)

# Liaison UART : caractéristiques

- Liaison série
- Asynchrone
- Full duplex
- Pair à pair




- Pas de configuration automatique
  - (Mais certains récepteurs autodétectent la vitesse de transmission)

# Liaison UART : dénomination

- UART fait référence au composant de sérialisation / dé-sérialisation
  - Parler de *liaison UART* est un abus de langage
- Liaison série
  - Par opposition à l'autre liaison (parallèle) disponible en standard sur les PCs (jusque début années 2000)
  - aujourd'hui TOUTES les liaisons d'un PC sont séries
- Liaison RS-232
  - La norme RS-232 utilise le principe de l'UART mais l'associe à des contraintes d'implémentation
    - Niveaux de tension +12 (0 logique) / -12 (1 logique)
    - Vitesses de transfert (mutiples ou sous-multiples de 9600 bps)
    - Longueur maximale de câble
  - Aujourd'hui, la plupart des liaisons sont en CMOS 3.3V à 115200bps
    - Incompatible avec la norme RS-232

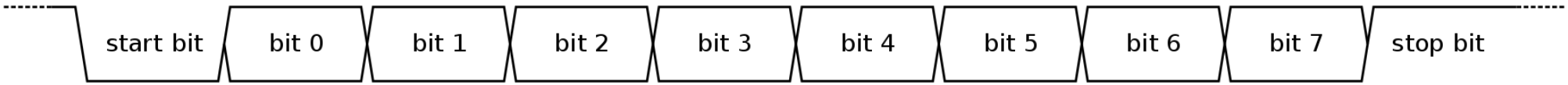
# La liaison UART : principe

- L'émetteur et le récepteur sont configurés pour utiliser les mêmes paramètres, notamment:
  - Vitesse d'envoi des bits (durée de chaque bit)
  - Le nombre de bits
  - d'autres paramètres qu'ons verra plus tard…
- Une ligne est utilisée pour envoyer, une autre pour recevoir → pas de gestion d'arbitrage

- Au repos (pas d'envoi), la ligne est au '1' logique.
  - Un bit '0' est envoyé (start bit) avant chaque donnée
  - La donnée est envoyée (LSB en premier)
  - Un bit '1' est envoyé (stop bit)


Source : Wikipedia – UART timing Diagram

# La liaison UART : principe




- En réception:
  - Un front descendant sur la ligne marque le début d'un mot
  - Début et fréquence connus → décodage *facile*
  - On vérifie la fin du mot (stop bit = 1)
  - Prêt pour le mot suivant

# La liaison UART : paramètres

- Vitesse (souvent 9600 ou 115200)
  - Définie en fonction de l'horloge système pour un µC
- Nombre de bits par mot ( de 5 à 9, quasiment toujours 8)
  - Parfois fixe poru les µC
- Présence d'un bit de parité (paire – impaire – non)
- Nombre de bits de stop (1 – 1,5 – 2, quasiment toujours 1)
- Contrôle de flux (matériel – Xon/Xoff – aucun )
  - Rarement utilisé, les µC traitent souvent les données à la volée
  - Si contrôle logiciel (Xon/Xoff), le composant doit être réglé sur "aucun"

- Un périphérique distant peut autodétecter la vitesse
  - Esclave uniquement
  - Éviter (problèmes au RESET)

# UART : usage en µC

- Souvent UART est une sous-fonction d'un module plus complet :
  - PIC : (gère aussi transmission synchrone / interfaces IR)
- Une série de registres pour définir les paramètres de la liaison
- Un registre de statut, qui contient au moins
  - 1 bit de réception de donnée
  - 1 bit lié à l'envoi (canal libre et/ou place disponible en buffer)
- Un registre de réception
  - Il contient le dernier caractère reçu
  - Si on le lit, le bit de réception passe à 0
- Un registre d'envoi
  - l'écriture d'une donnée provoque son envoi
  - Si on l'écrit, le bit d'envoi réagit en fonction

# UART : usage en µC

- Il y a rarement une FIFO intégrée (trop cher pour un µC)
  - S'il y a une FIFO, elle est petite
  - Sinon, en langage commercialement correct, on dit qu'il y a une fifo à 1 élément
  - Il faut surveiller quand on peut/doit lire ou écrire avant de la faire
  - Le matériel ne peut pas générer d'accès bloquant le soft en µC (il faut le faire à la main)

    (possible en techno FPGA, mais dangereux)

    => c'est la partie logicielle qui doit se débrouiller pour ne pas perdre de données
- Les rapports temporels sont énormes
  - PIC32 : écriture dans un registre = 0.01µs
  - PIC8 : écriture dans un registre = 1µs
  - UART : envoi/réception d'un caractère à 115200bps = 87µs
  - UART : envoi/réception d'un caractère à 9600bps = 1,04ms

# Format de données:

- Comment sont codées les données dans une liaison ?
  → en binaire, sans typage ni interprétation

- Pour des caractères :
  - Il suffit de copier les caractères
- Pour des valeurs entières
  - Il faut être au format *char* pour être sûr que la donnée sera sur 8 bits
  - Si on veut des valeurs qui ne tienne pas sur un char :
    - Il faut envoyer plusieurs mots pour coder la valeur
      → Il faut un protocole pour différencier les mots selon leur signification ( MSB / LSB )
- Pour afficher une valeur numérique sur le terminal:
  - Il faut d'abord la convertir en texte, puis l'envoyer

# Utilisation de l'UART

- TP5 : scrutation uniquement
  - "Bonjour le monde" de façon simple avec des fonctions d'attente
  - Difficulté : récupérer les infos de la doc pour utiliser le hard
- TP6 : ajout des Interruptions (sinon c'est pas drôle)
  - Réception en IT / envoi en scrutation
  - Réception en scrutation / envoi en IT
  - Les deux ensemble
  - Écriture d'un pilote de périphérique / bibliothèque d'utilisation
- TP7 : fin du TP6 / utilisation du DMA (pour les plus témérraires)