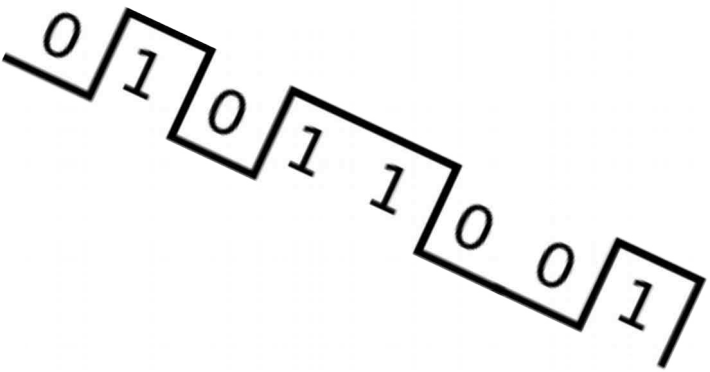
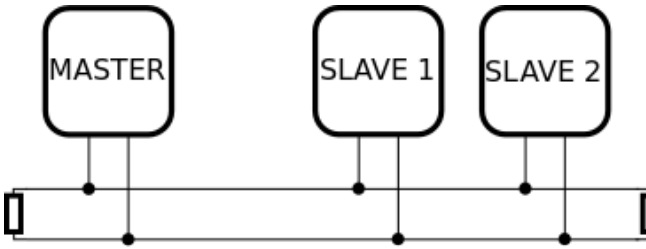
Module tronc commun : Réseau 2 et Bus de communication
Module spécialité AU : Ethernet Industriel

Seconde année Génie Électrique et Informatique Industrielle



Bertrand Vandeportaële
Jonathan Piat

Version 2021



Présentation

- Bertrand Vandeportaële et Jonathan Piat, Enseignants Chercheurs
- Robotique au LAAS
- WIKI https://bvdp.inetdoc.net/
  - Polycopiés de cours
  - Sujets de Travaux Pratiques
  - Ressources Documentaires

Au menu...

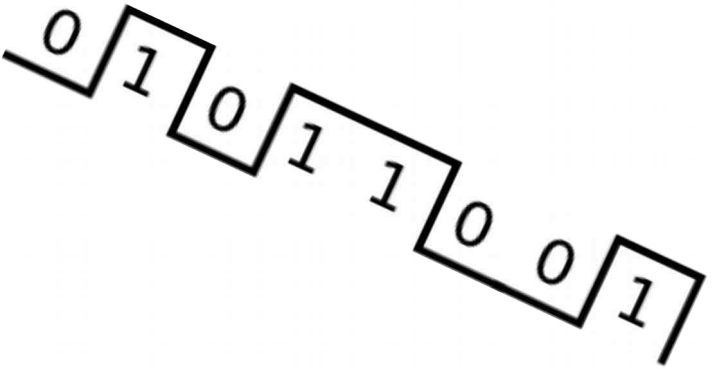
- Module tronc commun
  - Réseau 2: Mise en oeuvre d'une communication Ethernet entre un microcontrôleur PIC32 et un PC
    - Programmation Orientée Objet
    - Développement d'interface graphique avec QTCreator
    - Environnement MPLABX
  - Bus de communication:
    - Généralités sur les Bus
    - Bus séries : RS232, I2C, SPI
    - Bus parallèles : VME...
    - Mise en œuvre en TP sur plateforme Arduino
  - Evaluation :
    - TP + QCM (Documents autorisés)

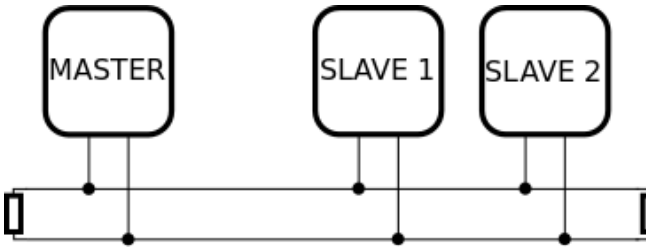
Au menu...

- Module spécialité AU :
  - Ethernet Industriel
    - Pas d’Ethernet du tout en fait...
    - I2C, SPI, Onewire
    - Protocoles (NMEA pour GPS, commandes AT pour MODEM)
    - Sans fils (infrarouge et radio)
    - Mise en œuvre en TP sur plateforme Arduino
  - Évaluation :
    - TP + Partiel Standard (Documents autorisés)

Généralités sur les Bus de communication:

Signaux
Topologie
Simplex/Duplex
Débits
Série/parallèle
Synchrone/asynchrone
Connectique et cablage
Couches de communication
FIFO
Détection d'erreur
Arbitrage de bus
Périphérique intégré/émulé
Perturbation/immunité



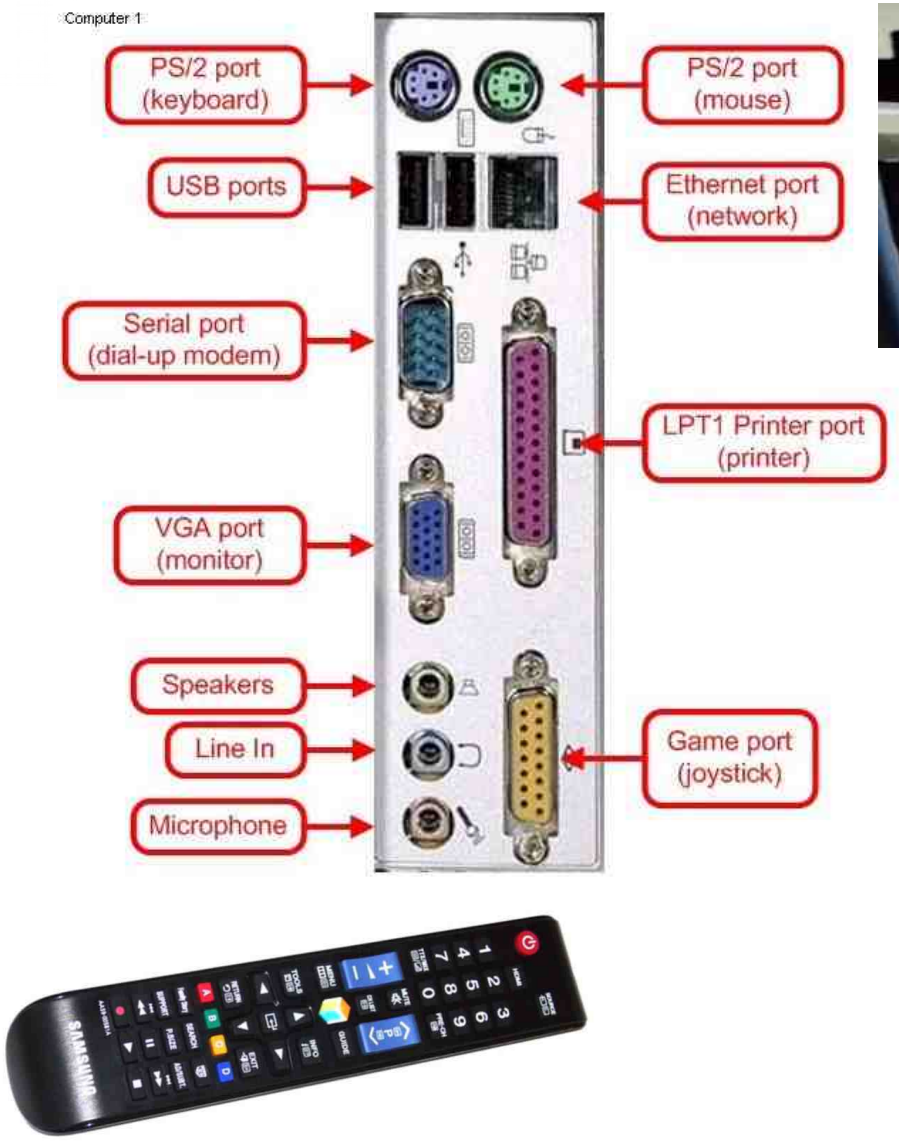


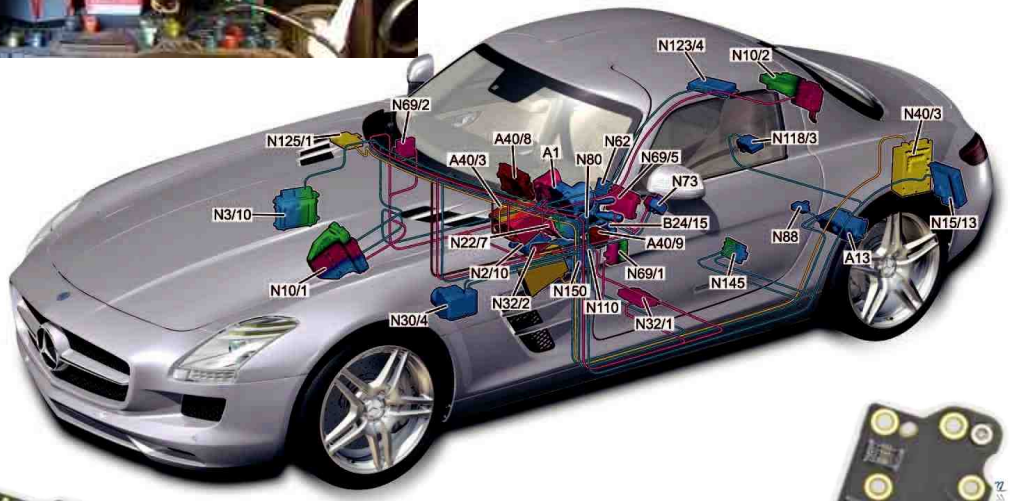
Définition d'un bus de communication

- **Bus matériel** : Support physique pour l'échange de données entre un ou plusieurs composants.
- **Bus Logiciel** : Support logique pour l'échange de données entre des composants logiciels.

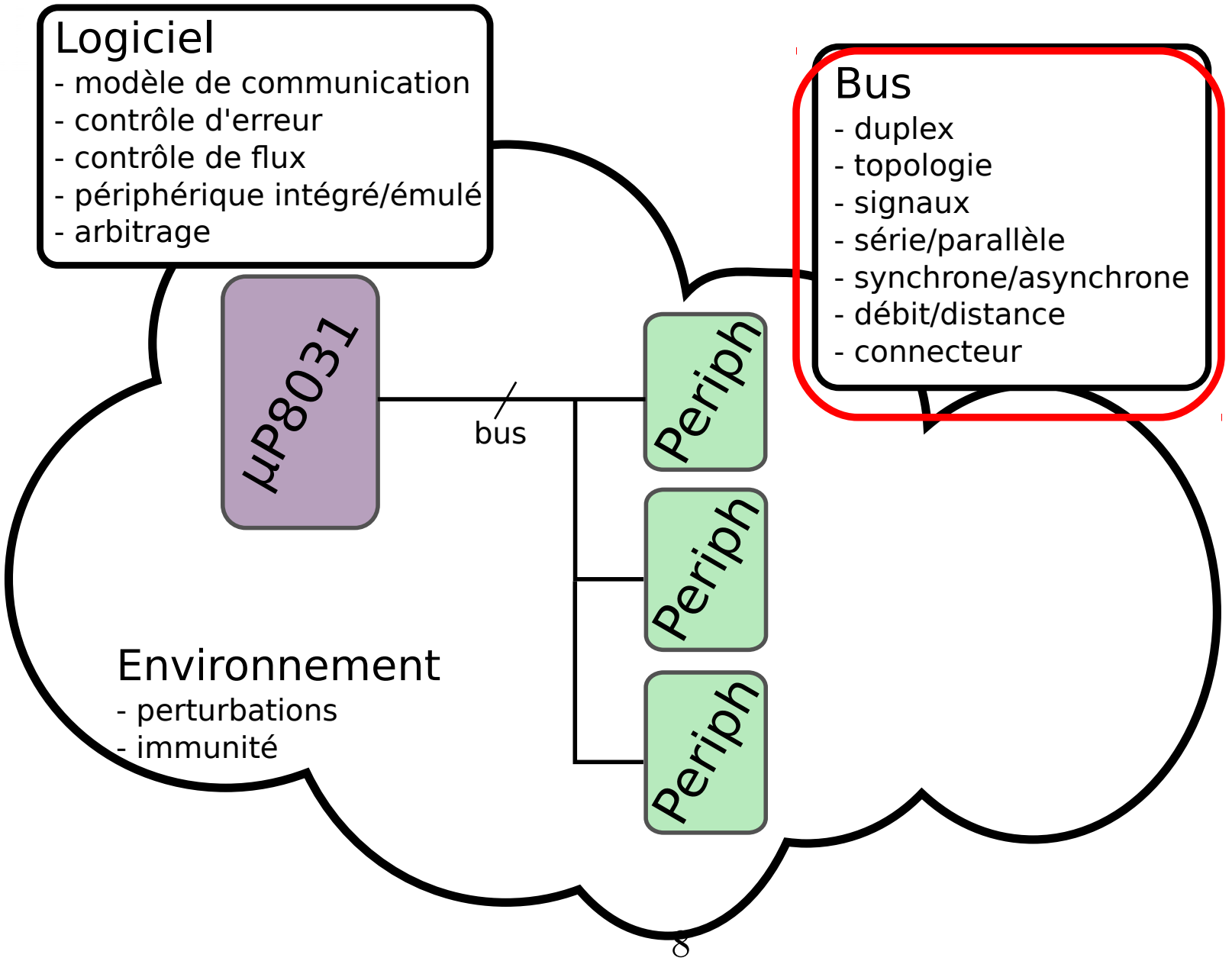
- Uniformisation grâce à l’abstraction :
  - par exemple la communication inter-processus via Sockets UDP/TCP sur IP de boucle locale utilise l’API Réseau

Les bus matériels qui nous entourent





Sommaire


Logiciel
- modèle de communication
- contrôle d'erreur
- contrôle de flux
- périphérique intégré/émulé
- arbitrage
Bus
- duplex
- topologie
- signaux
- série/parallèle
- synchrone/asynchrone
- débit/distance
- connecteur
µP8031
bus
Periph
Periph
Periph
Environnement
- perturbations
- immunité

Signaux

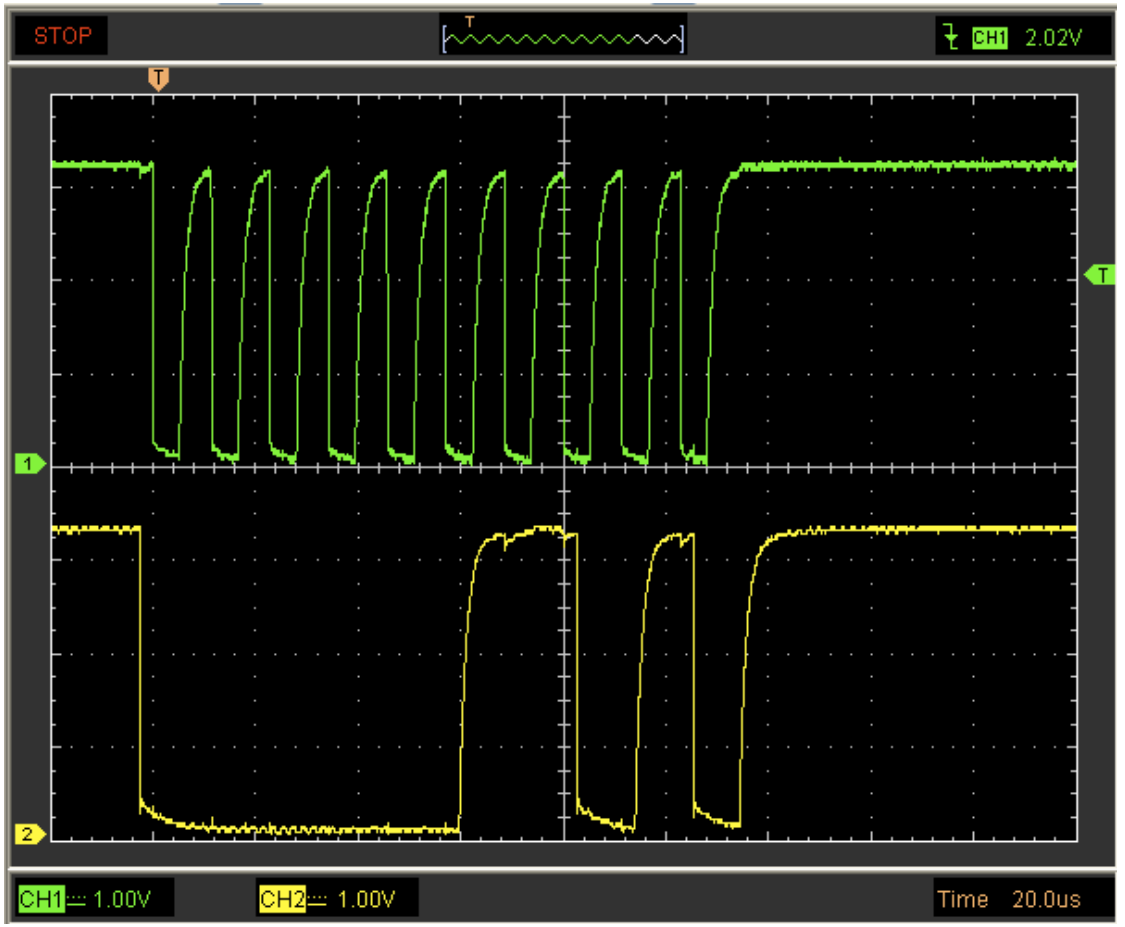
- Signaux de données:
  - Signaux véhiculant les données de la communication
- Signaux de contrôle:
  - Signaux permettant la synchronisation entre les acteurs (composants émetteurs et/ou récepteurs) du bus

Signaux

- Exemple : Transaction sur un bus I2C

Signal d'horloge

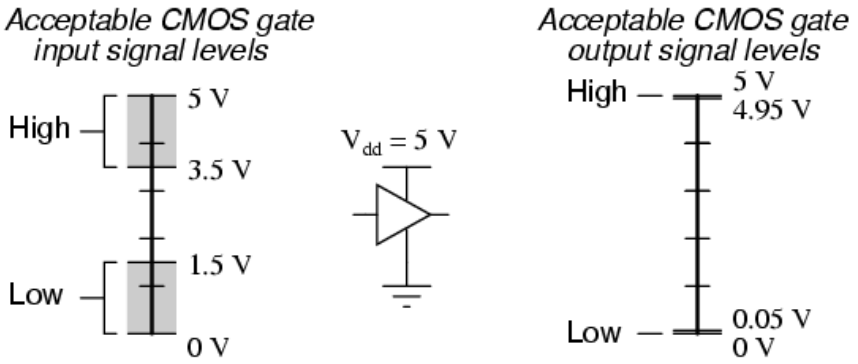
Signal de donnée



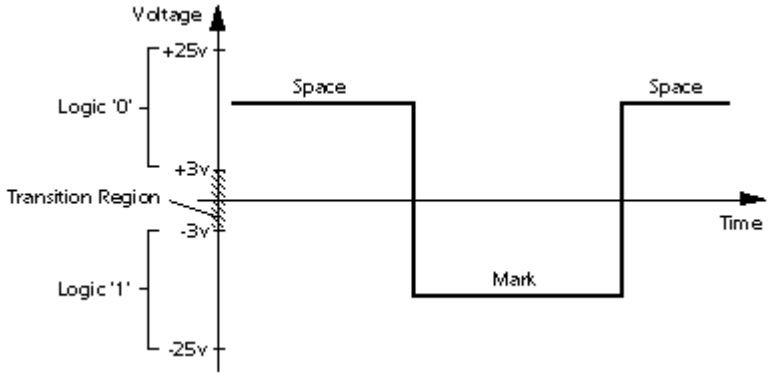
Signaux

- Caractéristiques électriques
  - Niveaux de tension, exemples :

CMOS



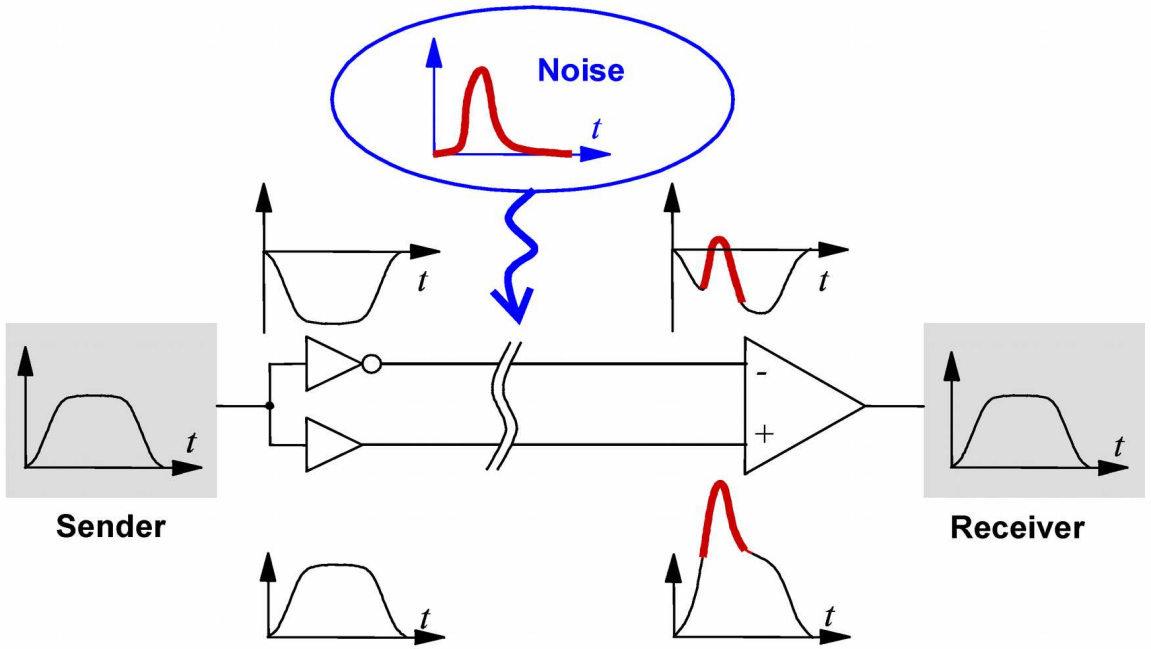
RS232

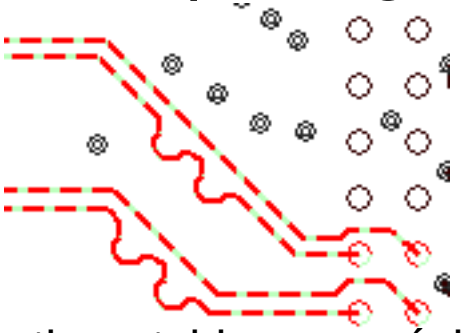


  - Courant

Signaux

- Mode de transmission
  - Asymétrique : 1 conducteur par signal
  - Symétrique/différentiel : 2 conducteurs par signal

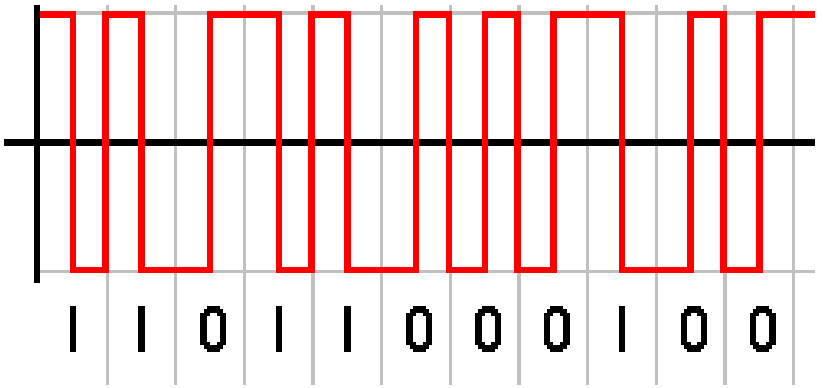




length matching pour éviter le déphasage

Signaux

- Représentation des symboles
  - NRZ : Non Return to Zero
    - un niveau pour chaque état logique
  - Codage de Manchester
    - une séquence pour chaque état logique
    - Le signal généré contient l'horloge



Signaux

- Que se passe-t-il quand deux hôtes imposent deux états différents sur le bus ?

- Gestion possible des conflits par 2 états :
  - État récessif : pouvant être altéré par un état dominant. (exemple : lumière éteinte)
  - État dominant : ne pouvant être altéré. L'acteur imposant un état dominant, prend le contrôle du bus. (exemple : lumière allumée)

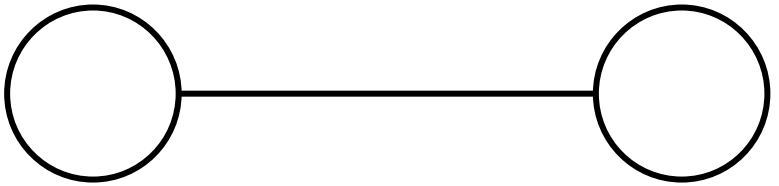
Notions de maître-esclave et pair à pair

- Dans une communication maître-esclave (master-slave)
  - Un maître peut initier une communication
  - Un esclave ne peut que répondre à une sollicitation d'un maître
- Dans une communication en pair à pair (peer to peer)
  - Les différents composants ont le même rôle, chacun peut initier la communication

Topologie

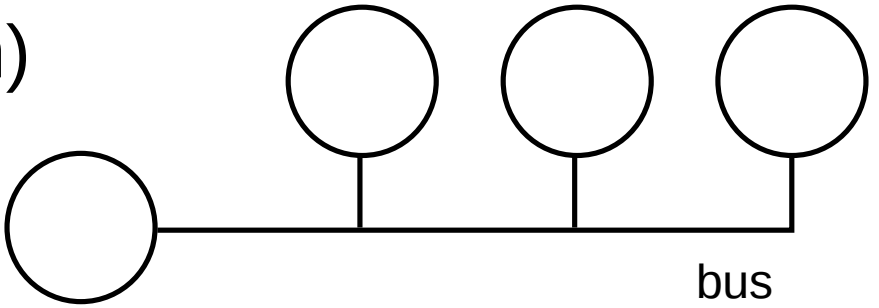
- Définit les interconnexions possibles entre les composants sur le bus

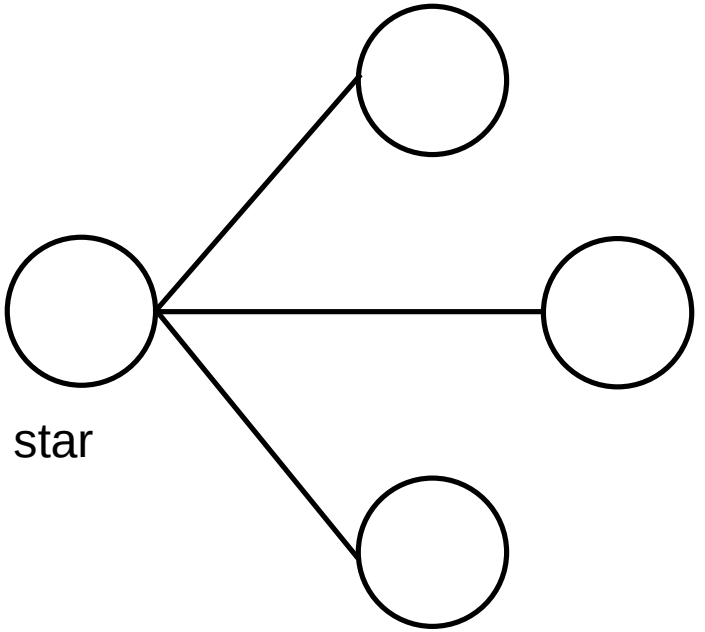
- Topologie Point à Point
  - Communication entre deux composants
  - Communication pair à pair ou maître-esclave



Topologie

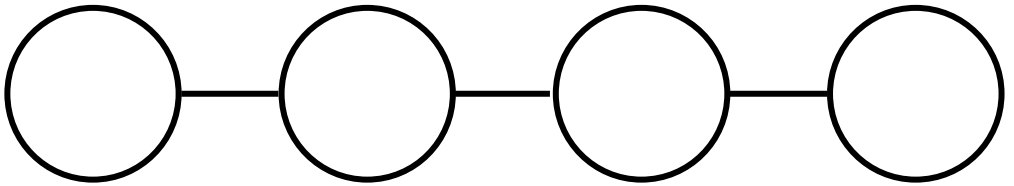
- Topologie Point → Multi-points
  - Communication entre plus de deux composants
  - Adressage (et/ou diffusion)
  - Communications:
    - Maître → Esclaves
    - Multi-maîtres → Esclaves
    - Pairs à Pairs





Topologie

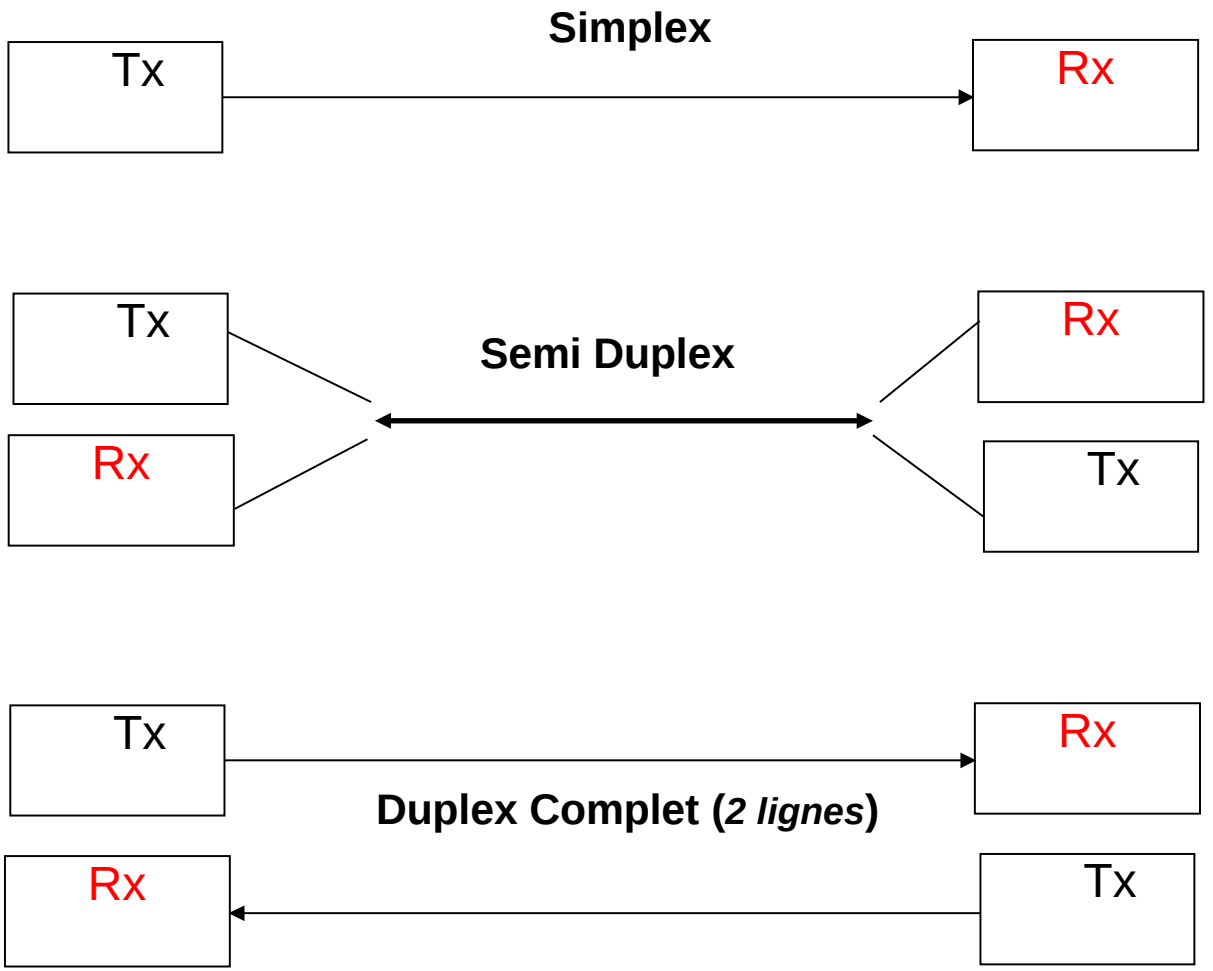
- Topologie Daisy-Chain
  - Communication entre plusieurs composants mis en cascade sur le bus.
  - Chaque composant transmet la donnée à son voisin direct si elle ne lui est pas destinée.
  - Danger : si la chaîne est rompue, tous les composant en aval ne peuvent plus communiquer.
  - L’adressage peut être fait à partir de l’ordre des composants dans la chaîne.



Simplex/Duplex

- Simplex : Communication dans un seul sens.
- Half-duplex : Changement du sens de communication au cours du temps. Un seul sens de communication à la fois.
- Full-duplex : Communication simultanée dans les deux sens.

Simplex/Duplex

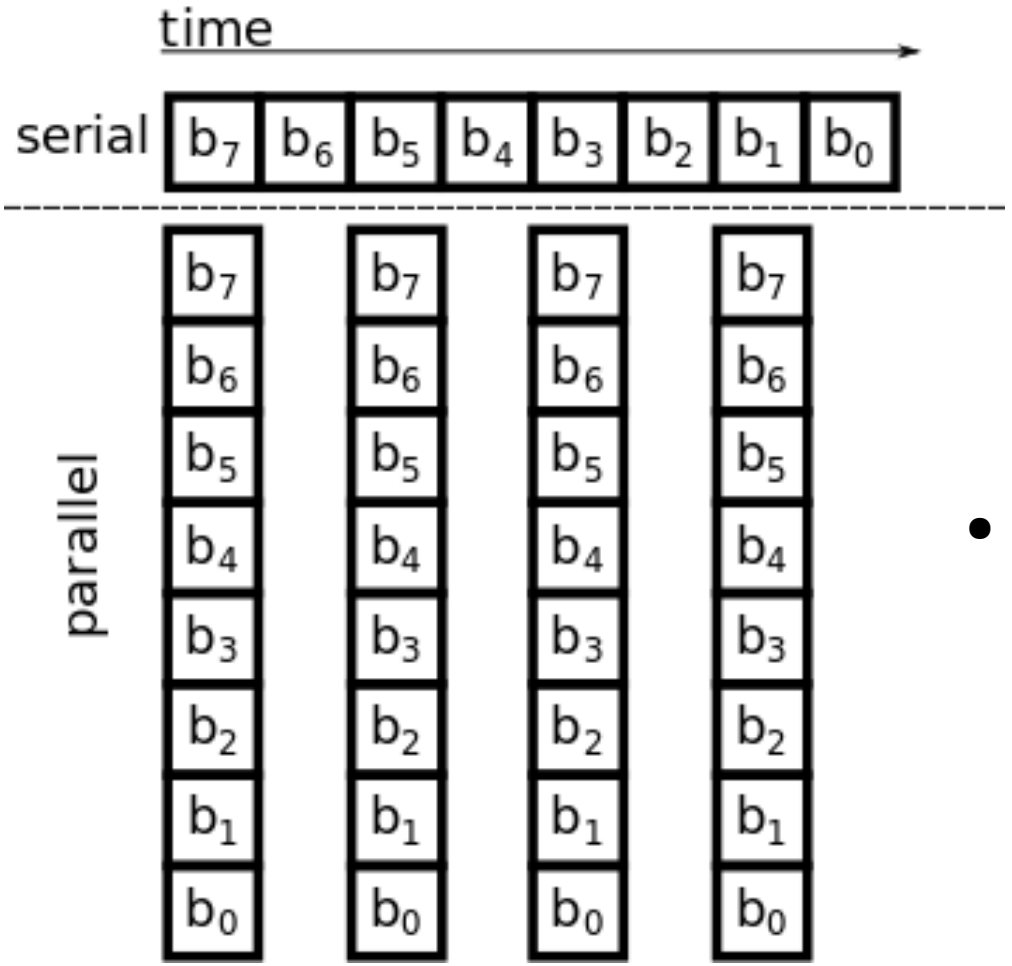


Débit, Vitesse de communication

- Bit-rate : Nombre de bits de données échangés par seconde bit/s (bs)
- Data-rate : Nombre d'octets envoyés par seconde os en français, Bps (Byte per second) en anglais
- Baud-rate : Nombre de symboles échangés par seconde (bds). Un symbole peut coder plusieurs combinaisons de bits.

$$\text{Bitrate} = \text{baudrate} * \ln_2(\text{nombre de symboles})$$

Bus Parallèle ou Série



- Bus Série : le mot de donnée (octet) est découpé en une série de symboles (bit) transmis les un après les autres
- Bus parallèle : les données sont organisées en mot (groupe de bits) transmis simultanément.

Bus Série

- Avantages :
  -nombre de conducteurs réduit
  -synchronisation des signaux plus facile
  -distance de communication plus grande
- Inconvénients :
  - vitesse de transmission (dans certains cas)
    - Possibilité de mise en parallèle de plusieurs bus série (ex : PCIe 16X)
    - SATA vs PATA
    - Bus mémoire de processeur est parallèle
  - coût matériel de la sérialisation/désérialisation
  - latence

Bus Série Synchrone

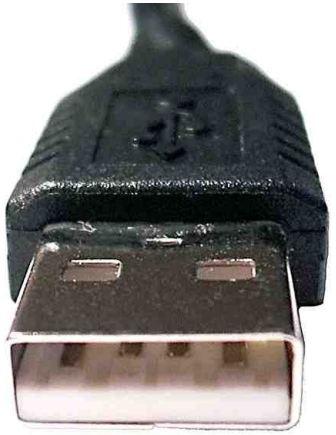
- Le bus transmet les symboles (bits) synchrones **avec une horloge**, elle aussi disponible sur le bus.
- Avantages:
  - Vitesse du bus (pas de reconstitution de la synchronisation)
- Inconvénients:
  - Câblage (nombre de conducteurs et distance)
  - Sensible au bruit (risque de fronts parasites sur le signal d'horloge)

Bus Série Asynchrone

- La bus ne véhicule pas de signaux de synchronisation
- La **synchronisation est récupérée par les données**
- Avantage :
  - Moins de conducteurs pour former le bus.
  - Moins sensible au bruit.
- Inconvénients :
  - Vitesse du bus limitée

Connectique

- Permet la connexion des signaux du bus vers les composants du bus
- Connecteur femelle est le réceptacle du connecteur mâle

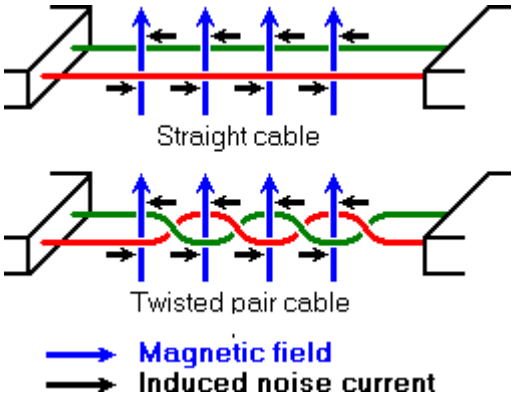


Connectique

- Assure la compatibilité électrique
- Définie :
  - Soit par le standard (Ex:USB)
  - Soit par l'application (Ex: OBD pour bus CAN)

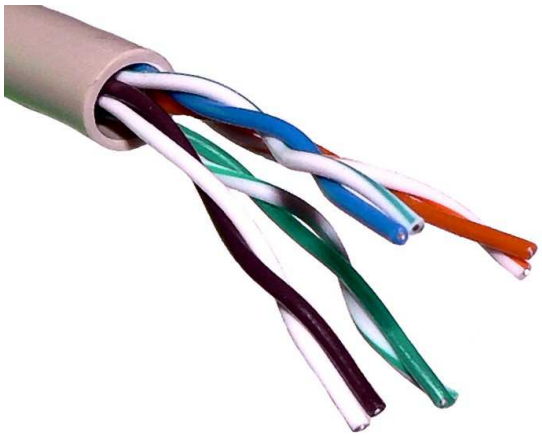
Câblage

- Assure la compatibilité électrique avec le mode de transmission (Ex: paire torsadée pour signaux différentiels)
- Assure l'immunité aux perturbations extérieures (blindage)
- Exemple de câble torsadé pour une paire différentielle :
  - Les perturbations électromagnétiques sur les différentes portions du câble se compensent



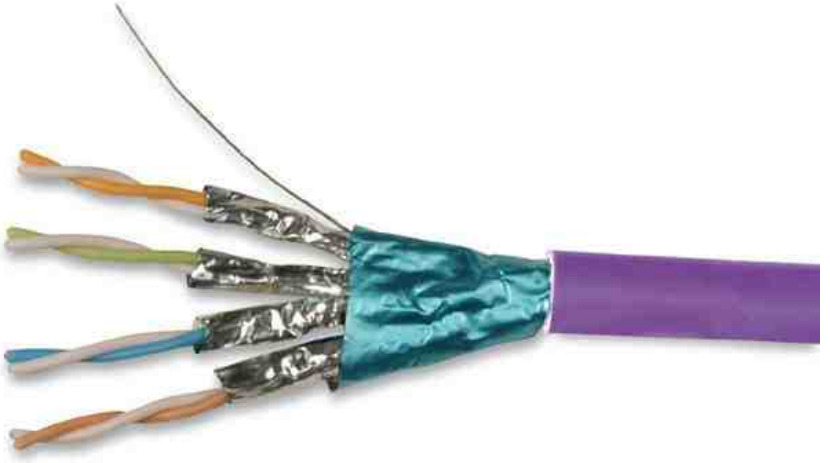
Câblage

- Exemple de câbles utilisés pour le réseau Ethernet



UTP
(unshielded twisted pair)

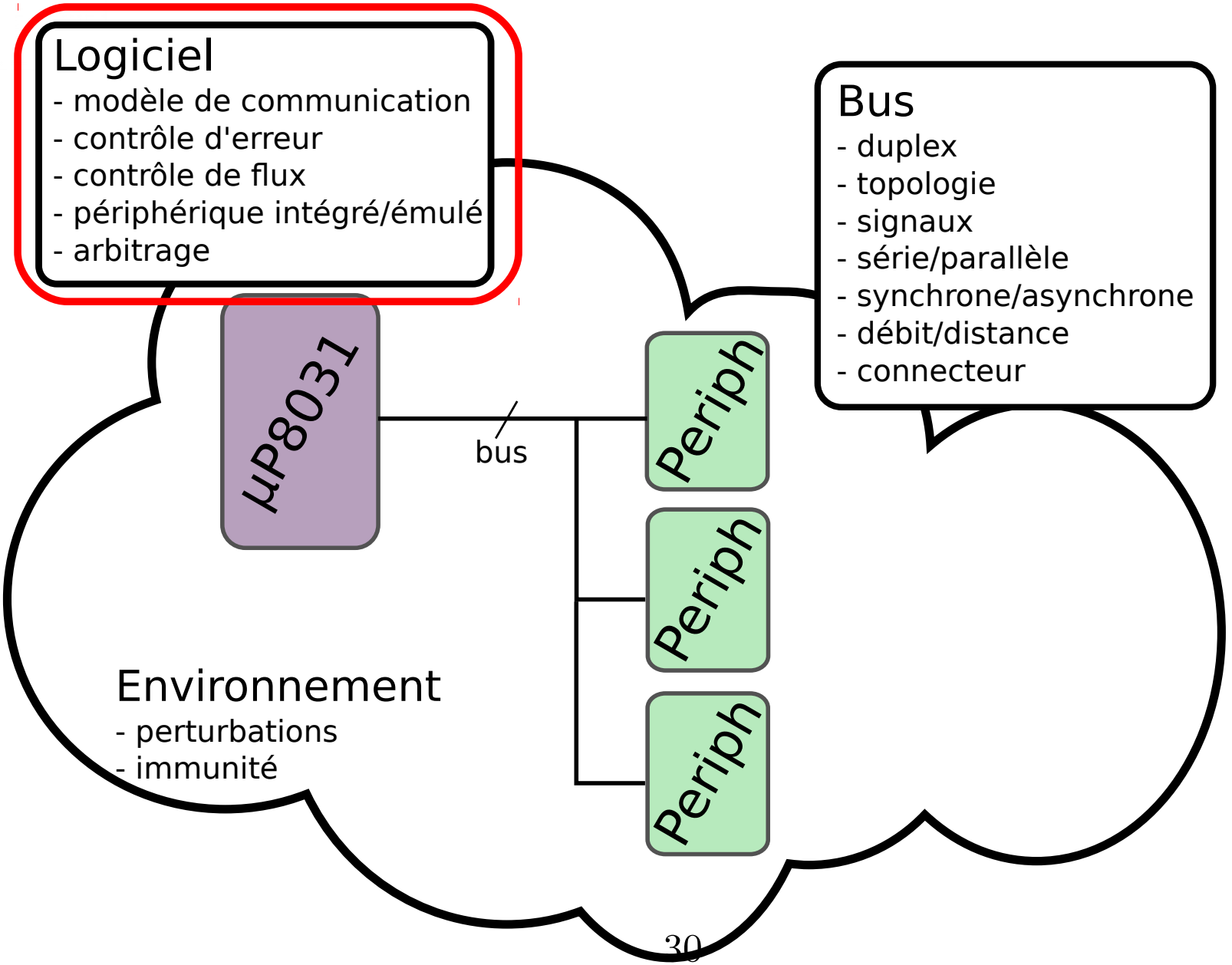
FTP (foil twisted pair): feuillard



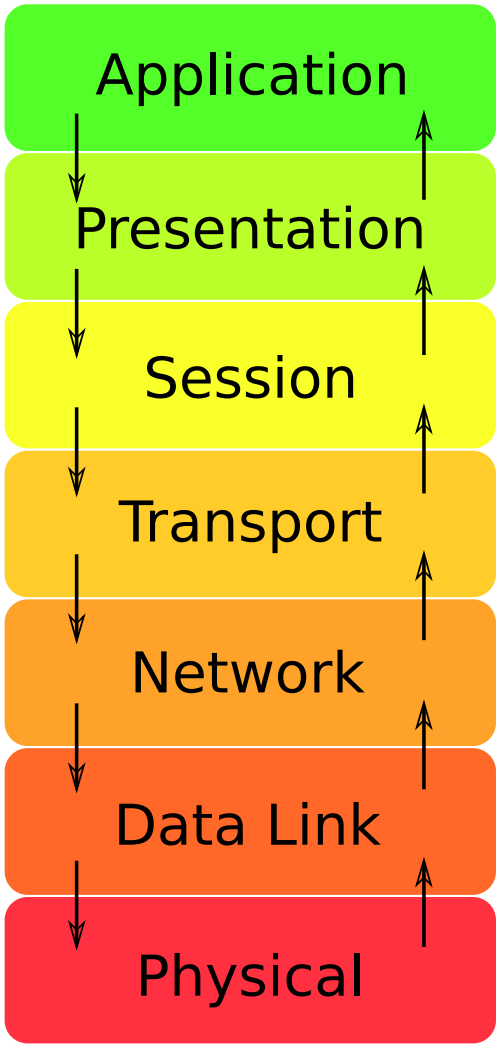
SFTP Cat7
(shielded foil twisted pair)

Catégorie du câble	Bande passante
CAT5e	100Mhz
CAT6	250Mhz
CAT6a	500Mhz
CAT7	600Mhz

Sommaire

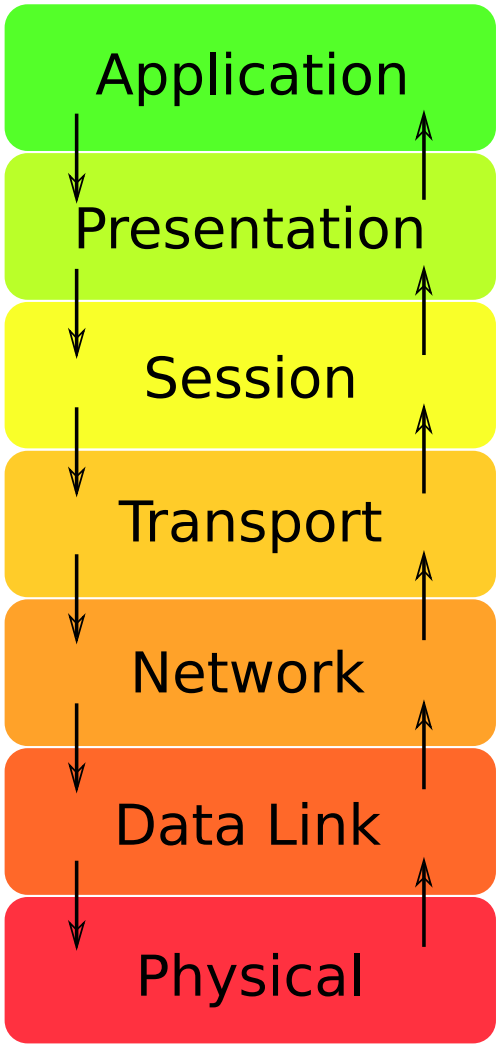

Logiciel
- modèle de communication
- contrôle d'erreur
- contrôle de flux
- périphérique intégré/émulé
- arbitrage
Bus
- duplex
- topologie
- signaux
- série/parallèle
- synchrone/asynchrone
- débit/distance
- connecteur
µP8031
bus
Periph
Periph
Periph
Environnement
- perturbations
- immunité

Couches de communication



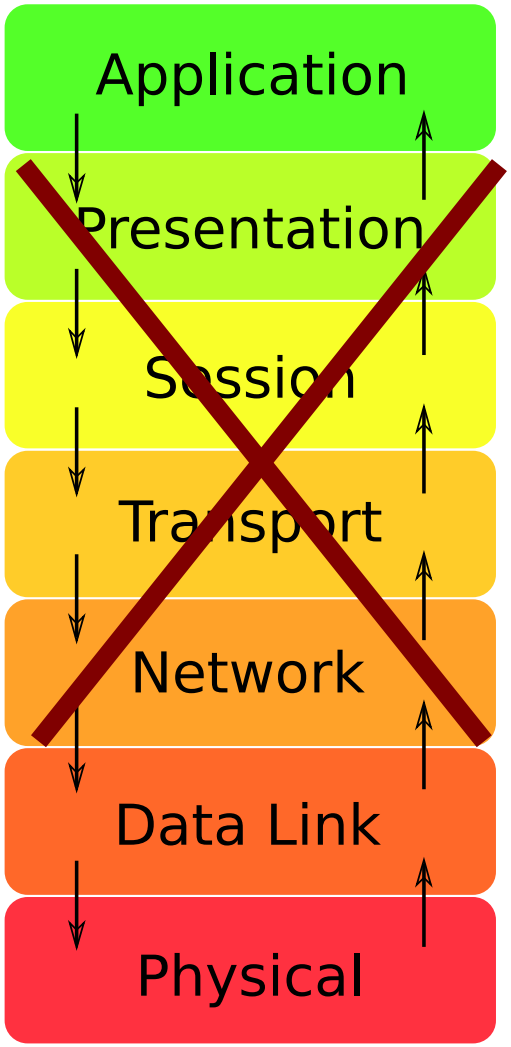
- Division en couches
- Communication entre les couches par primitives
- Unité de donnée associée à chaque couche (PDU)
- Permet la structuration des développements

Couches de communication



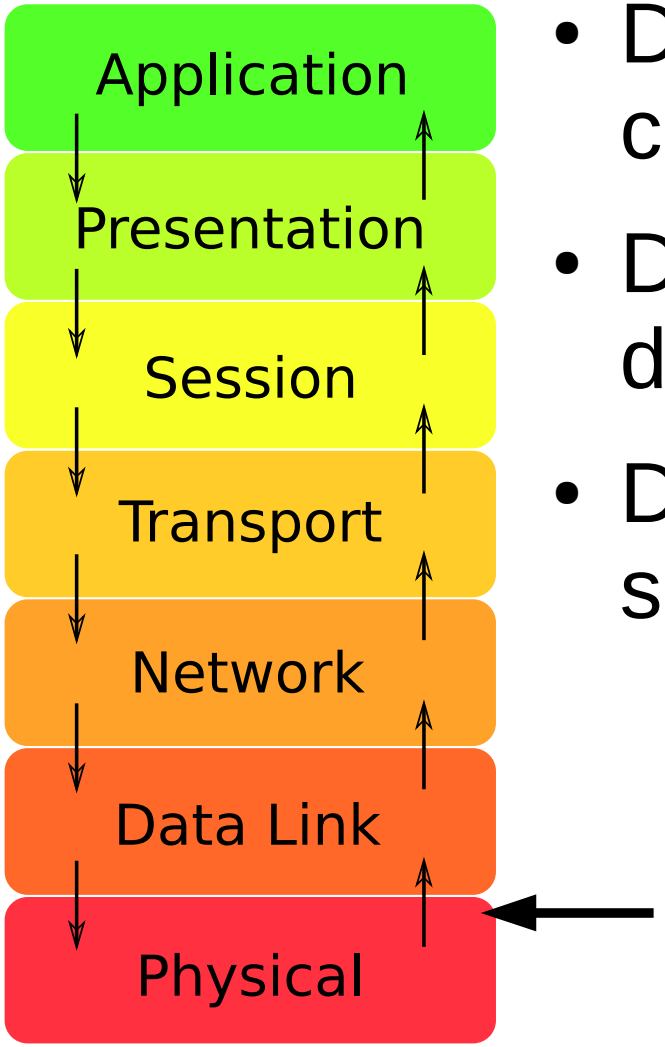
- Notion de librairie : Abstraction d'un périphérique à un certain niveau
- Ex : Librairie fournie par le fabricant d'un micro-contrôleur pour les périphériques intégrés

Couches de communication



- En pratique pour la plupart des bus que nous allons voir, 3 couches utilisées seulement

Couche Physique

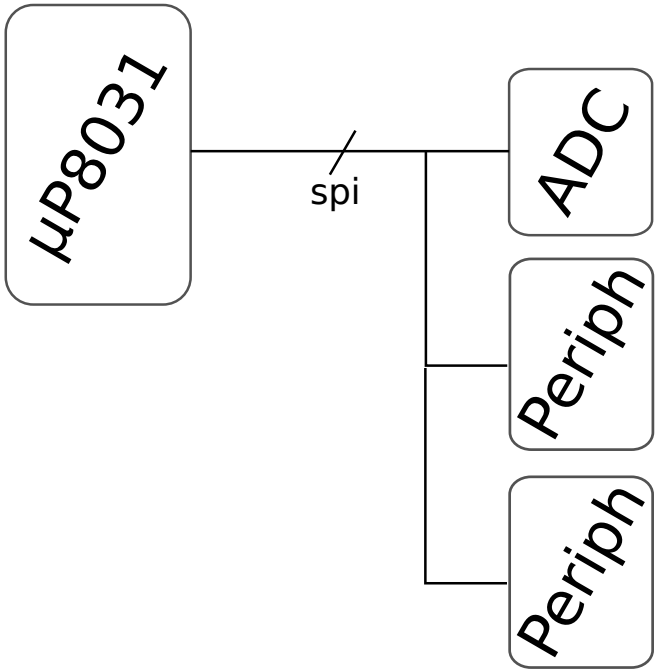


- Définit l'interface de communication sur le média
- Définit le codage des bits de donnée sur le bus
- Définit les topologies de bus supportées

Couche Physique

- Exemple de fonctions d'une librairie :
  - open_<péripherique>()
  - send_byte_<périphérique>(uchar byte)
  - uchar rcv_byte_<périphérique>(void)
  - uchar transfer_byte_<périphérique>(uchar byte)
  - close_<péripherique>()

Couche Physique



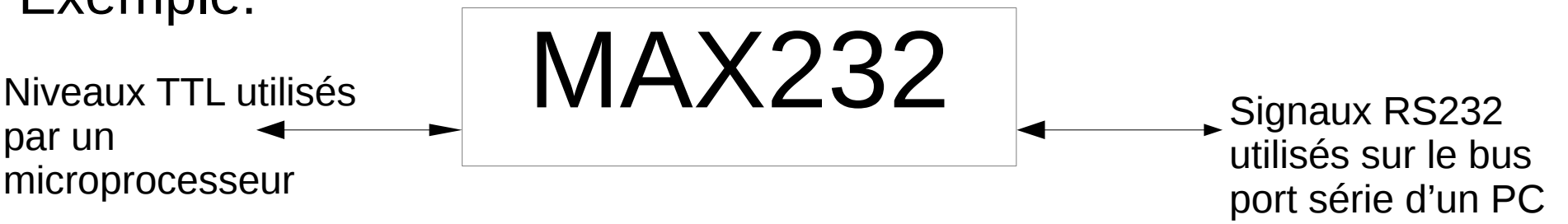
Physical

uchar transfer_byte_spi(uchar send)

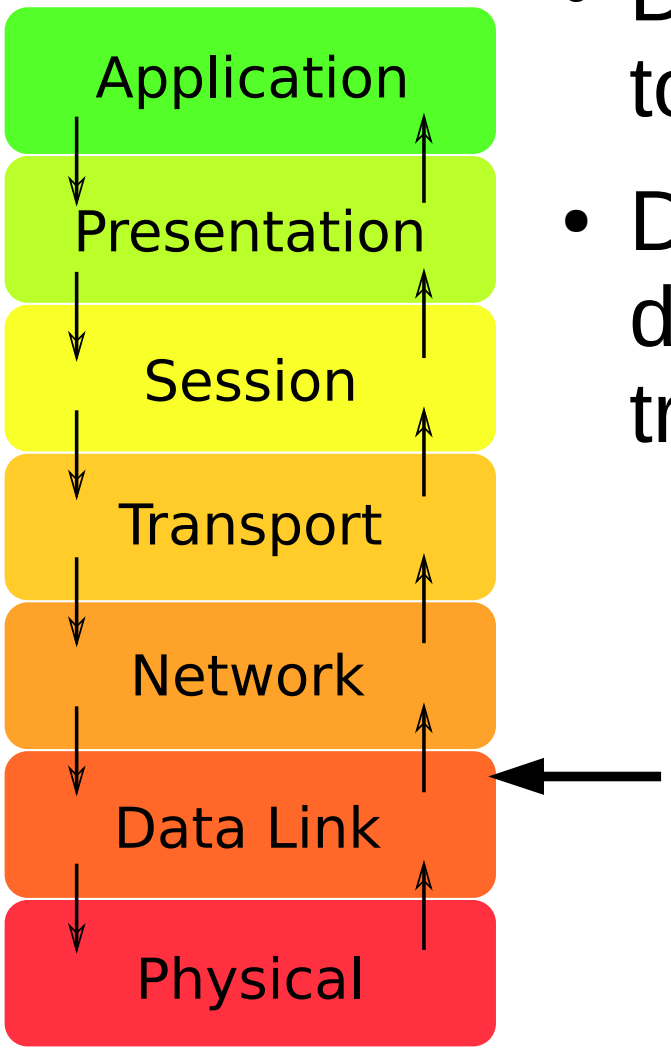
Couche Physique

- Un composant électrique externe peut être requis pour adapter les signaux logiques utilisant certains niveaux et les signaux physiques utilisés sur le bus, ce composant est appelé le PHY

Exemple:



Couche Liaison

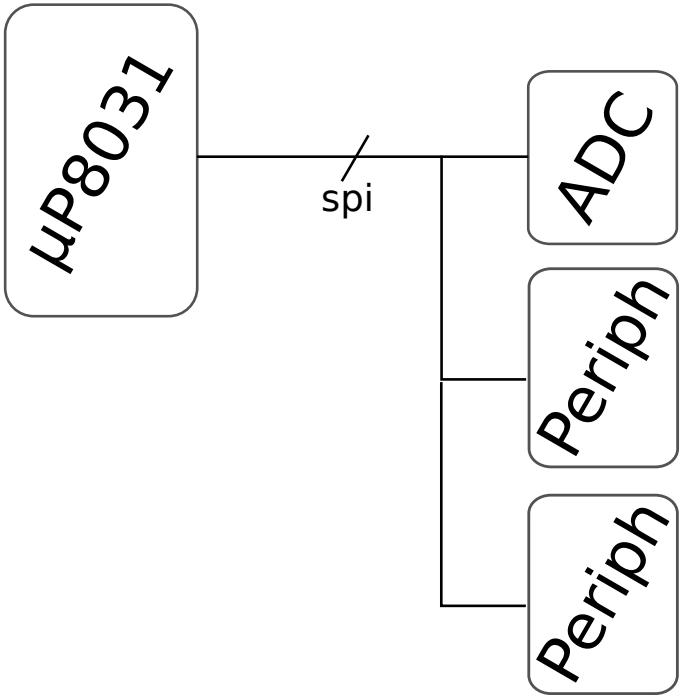


- Définit l'adressage dans le cas des topologies point → multi-points
- Définit le formatage et le découpage des données transmises sur le bus

Couche Liaison

- Exemple de fonctions d’une librairie :
  - send_data_<périphérique>(uchar * data, unsigned int addr, unsigned int nb)
  - uchar rcv_data<périphérique>(uchar * data, unsigned int addr, unsigned int nb)
  - uchar transfer_data<périphérique>(unsigned int addr, uchar * snd, uchar * rcv, unsigned int nb)

Couche Liaison



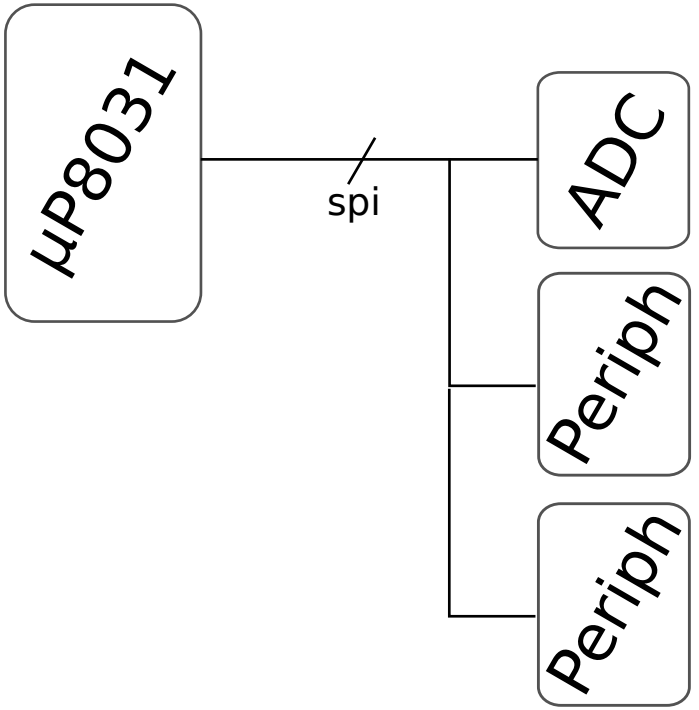
Data Link

```
uchar transfer_buffer_spi(uchar addr,
uchar * send, uchar * rcv, uint nb)
```

Physical

```
uchar transfer_byte_spi(uchar send)
```

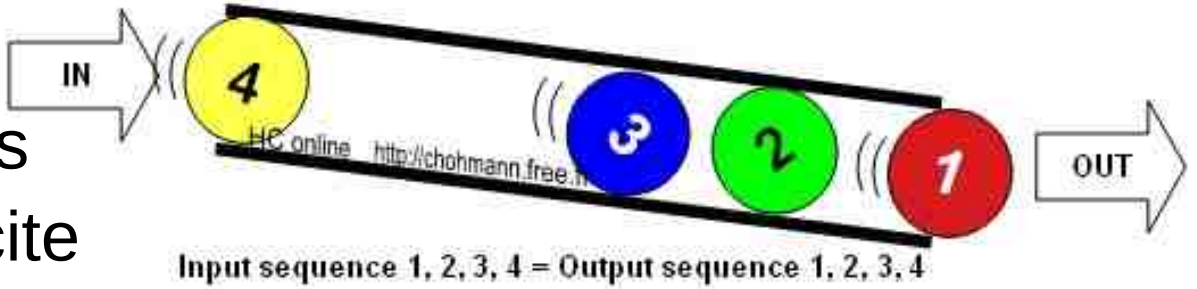
Couche Application



Application	readADCChannel(uchar nb, uint * val)
Data Link	uchar transfer_buffer_spi(uchar addr, uchar * send, uchar * rcv, uint nb)
Physical	uchar transfer_byte_spi(uchar send)

Modèle de communication

- First In First Out
  - File Matérielle ou Logicielle
  - Permet de relier deux domaines asynchrones
  - Synchronisation implicite
  - Buffer de taille fixe
  - Le producteur écrit des tokens (jetons) dans la FIFO
  - Le consommateur lit des tokens dans la FIFO
  - La lecture/écriture préserve la séquentialité des données
  - Un token peut être un octet (uchar) ou n'importe quel type de donnée, mais gestion de la FIFO plus simple si tous les tokens sont de même type



Modèle de communication

- FIFO: Exemple d'implémentation C

```
struct my_fifo{

        unsigned char buffer [FIFO_SIZE] ;

        unsigned int write_index ;

        unsigned int read_index;

        unsigned int nb_data_available ;

};


  void fifo_init(struct my_fifo * fif){
        fif->write_index  = 0 ;
        fif->read_index  = 0 ;
        fif->nb_data_available = 0 ;
  }
```

Modèle de communication

```
bit fifo_read(struct my_fifo *  fif, unsigned char * data) {
        if(fif->nb_data_available == 0) return 0 ;
        *data = fif->buffer[fif->read_index];
        fif->read_index = (fif->read_index + 1) % FIFO_SIZE;
        fif->nb_data_available--;
        return 1;
}
bit fifo_write(struct my_fifo * fif, const unsigned char c) {
        if (fif->nb_data_available <  FIFO_SIZE) {
                fif->buffer[fif->write_index] = c;
                fif->write_index = (fif->write_index + 1) % FIFO_SIZE;
                fif->nb_data_available ++;
                return 1;
        }
        return 0;
}
```

Modèle de communication

- Quand utiliser une FIFO ?
  - Producteur produit à une fréquence plus élevée que le consommateur (consomme) sur un intervalle de temps suffisamment court.
  - Permet de lisser les pics de production
- Taille de la FIFO ?
  - Connaître la fréquence d'écriture (push)
  - Connaître la fréquence de lecture (pop)
  - Connaître/Calculer la durée avant la perte de données

Erreur de transmission

- Taux d'erreur bit (Bit Error Rate) :
  - Caractérise la fiabilité de la transmission
- Techniques de détection d'erreur:
  - Couche physique : parité
  - Couche liaison : CRC, Checksum

Contrôle de parité

- Bit permettant de maintenir le nombre de '1', dans la donnée transmise, paire ou impaire
- Ajouté avant le bit de stop dans une trame série asynchrone
- 2 types possibles :
  - Parité pair (even) : bit à 1 si le nombre de '1' dans la donnée est impaire (ex : 10010100 → 1)

    permet d'obtenir un nombre pair de 1 dans la donnée+la parité
  - Parité impair (odd) : bit à 1 si le nombre de '1' dans la donnée est paire (ex : 10100011 → 1)

    permet d'obtenir un nombre impair de 1 dans la donnée+la parité

Parité, quelques exemples

Data	Count of '1' bits	Even	Odd
00000000	0	00000000**0**	00000000**1**
10100001	3	10100001**1**	10100001**0**
11010001	4	11010001**0**	11010001**1**
11111111	8	11111111**0**	11111111**1**

Contrôle d’erreur, CRC, Checksum...

- Utilisation d'un jeu d'octets réduit
  - Codage des données en ASCII, permet de faciliter la détection d'erreurs (exemple transmission de « 0x30,0x35 » pour 05)
  - Si réception d'un caractère non-ASCII, erreur ...
- Somme de contrôle :
  - Clé permettant de vérifier l'intégrité des données
  - Calculée par la somme arithmétique ou logique (xor) des données transmises
- Contrôle de Redondance Cyclique :
  - Calculé par des opérations de multiplications et xor avec une clé
  - Moins de risques de collisions (compensation des erreurs)

Arbitrage du bus

- Définit qui prend le contrôle du bus à un instant donné
- Maître-esclave:
  - Le maître gère l'arbitrage
- Pair à Pair:
  - Arbitrage par consensus
  - Arbitrage par règles de priorité
  - Arbitrage par composant externe

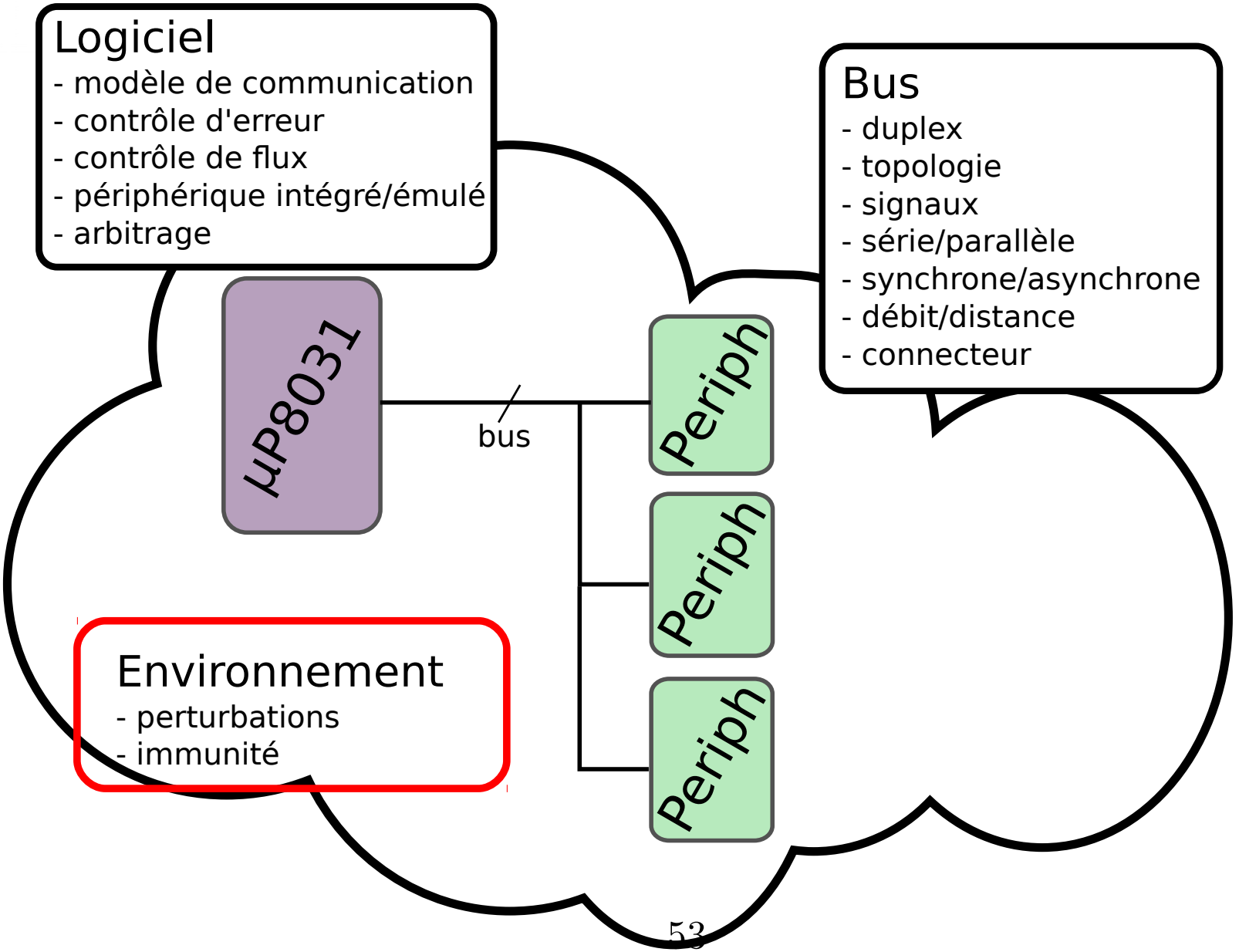
Périphérique Intégré (Matériel)

- Composant matériel disponible sur le µC
  - comme un timer, un PWM, une broche GPIO
  - Accessible via des registres
  - S’occupe d’une (+ou- grande) partie de la communication
- Avantages:
  - Temps parallèle: Le µC peut exécuter le programme en parallèle de la communication
  - Vitesse de communication indépendante de la fréquence d'exécution du µC
- Inconvénients:
  - Flexibilité réduite
  - Configuration fortement dépendante de la famille du µC
  - Nombre d'interfaces limité par construction
- Périphérique matériel externe
  - Par exemple l’UART 8250

Périphérique Émule (Logiciel)

- Composant logiciel exécuté par l'application
  - Les signaux de la communication sont générés à l‘aide de broches GPIO
- Avantages:
  - Indépendant de la disponibilité d'un périphérique matériel
  - Contrôle complet de l'interface (flexibilité)
- Inconvénients:
  - Gestion du “timing” en logiciel
  - Basse vitesse (dépendant de la fréquence d'exécution du µC)
  - Pas d'exécution en parallèle de la communication
  - Consomme de la mémoire programme et du temps CPU
    - Difficile à mettre en œuvre si le système doit remplir des contraintes temps réel fortes

Sommaire


Logiciel
- modèle de communication
- contrôle d'erreur
- contrôle de flux
- périphérique intégré/émulé
- arbitrage
Bus
- duplex
- topologie
- signaux
- série/parallèle
- synchrone/asynchrone
- débit/distance
- connecteur
µP8031
bus
Periph
Periph
Periph
Environnement
- perturbations
- immunité

Perturbations

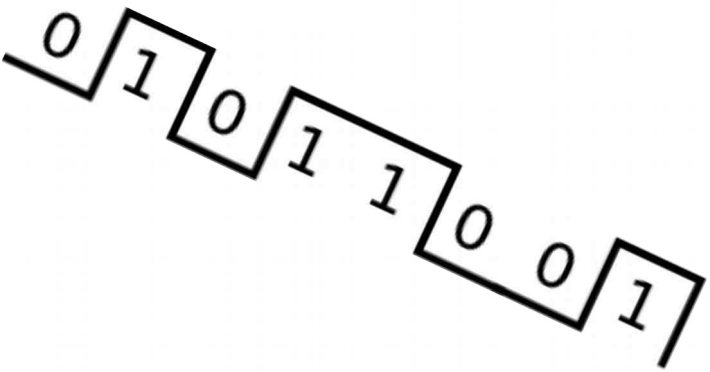
- Cross-Talk (diaphonie par induction électromagnétique)
- Rayonnement électromagnétique externe
  - Alimentation alternative
  - Machine-outils
  - Réseaux sans fils
- Longueur des lignes
- Impédance des lignes et terminaisons

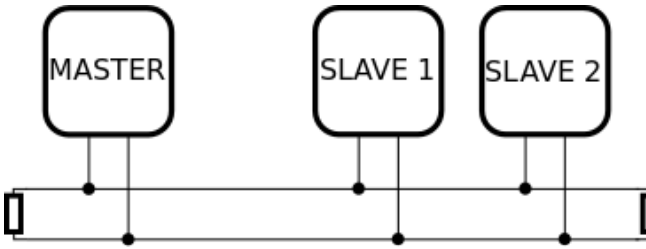
Immunité

- Blindage

- Choix d'un standard assurant la compatibilité électromagnétique avec l'environnement
- Utilisation d'un câble adapté pour la norme choisie

Bus Série asynchrone:
Bus RS232,RS422,RS485
Contrôle de flux
Bus USB



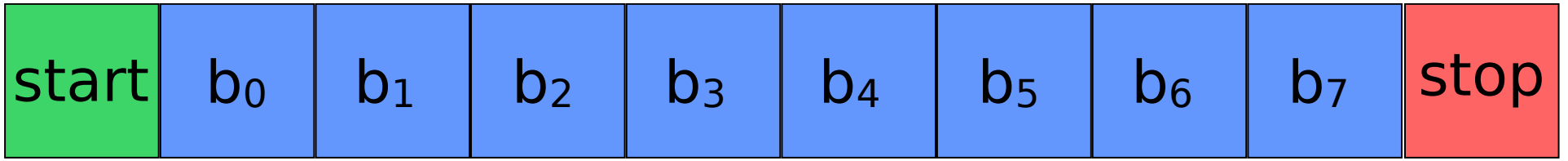


Bus Série asynchrone

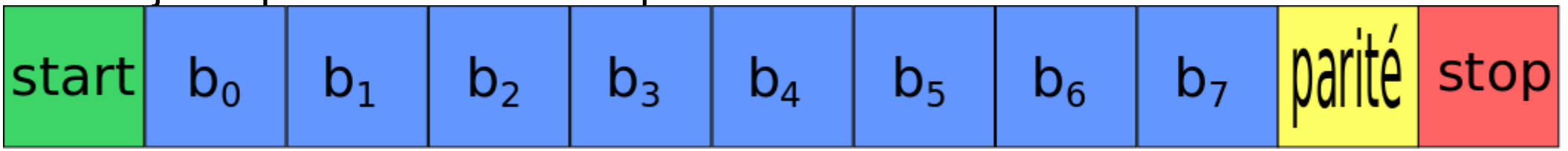
- Le bus ne véhicule pas de signaux de synchronisation
- La synchronisation est récupérée par les données
- Avantages :
  - Moins de conducteurs pour former le bus.
  - Moins sensible au bruit.
- Inconvénients :
  - La reconstitution de l'horloge à partir des données nécessite un traitement
  - Vitesse du bus limitée

Liaison UART

- UART : Universal Asynchronous Receiver Transmitter.
- Transmission des bits de données en série
  - nombre de bits configurable
  - Lsb first : transmission du bit (b minuscule) de poids faible d'abord
- Utilisation d'un bit de start et d'un (ou plusieurs) bit(s) de stop pour encadrer les symboles qui codent la donnée
  - Permet la resynchronisation à chaque début de trame



- Ajout optionnel d'un bit de parité



Liaison UART

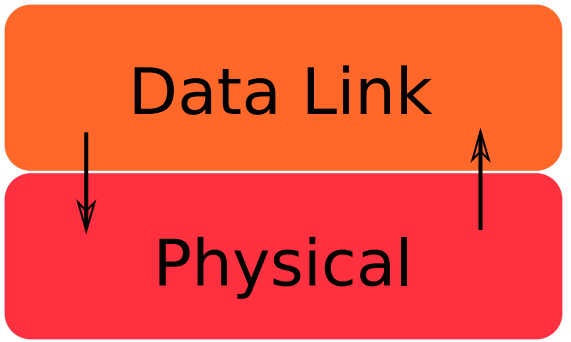
- Caractérisation, (triplet)
  - Un chiffre indiquant le nombre de bits de donnée utiles :
    - (5, 6, 7, 8, 9, 10 …)
  - Une lettre indiquant le type de parité utilisé :
    - N pas de parité
    - E parité paire (Even)
    - O parité impaire (Odd)
  - Un chiffre indiquant le nombre de bits de stop
    - (1, 1.5, 2...)
- Exemples :
  - 8N1 (8bits, pas de parité, 1 bit de stop)
  - 10E2 (10bits de données, parité paire, 2 bits de stop)

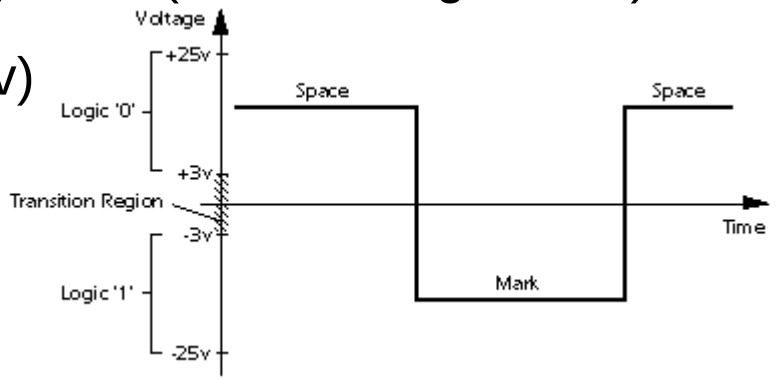
Liaison UART

- Pour liaison UART, Baudrate = Bitrate, nombre de bits transmis par seconde
- Notion de débit utile :
  - Synchronisation et détection d'erreur ont un coût :
  - débit bit utile=(Nb bits utiles/Nb bits transmis)*baudrate
    - 8N1 → 8/(8+1*start+1*stop+0*parité) → (8/10)*baudrate = 80% du baudrate
    - 8E2 → 8/(8+1*start+2*stop+1*parité) → (8/12)*baudrate = 66% du baudrate

RS232, Carte d'identité

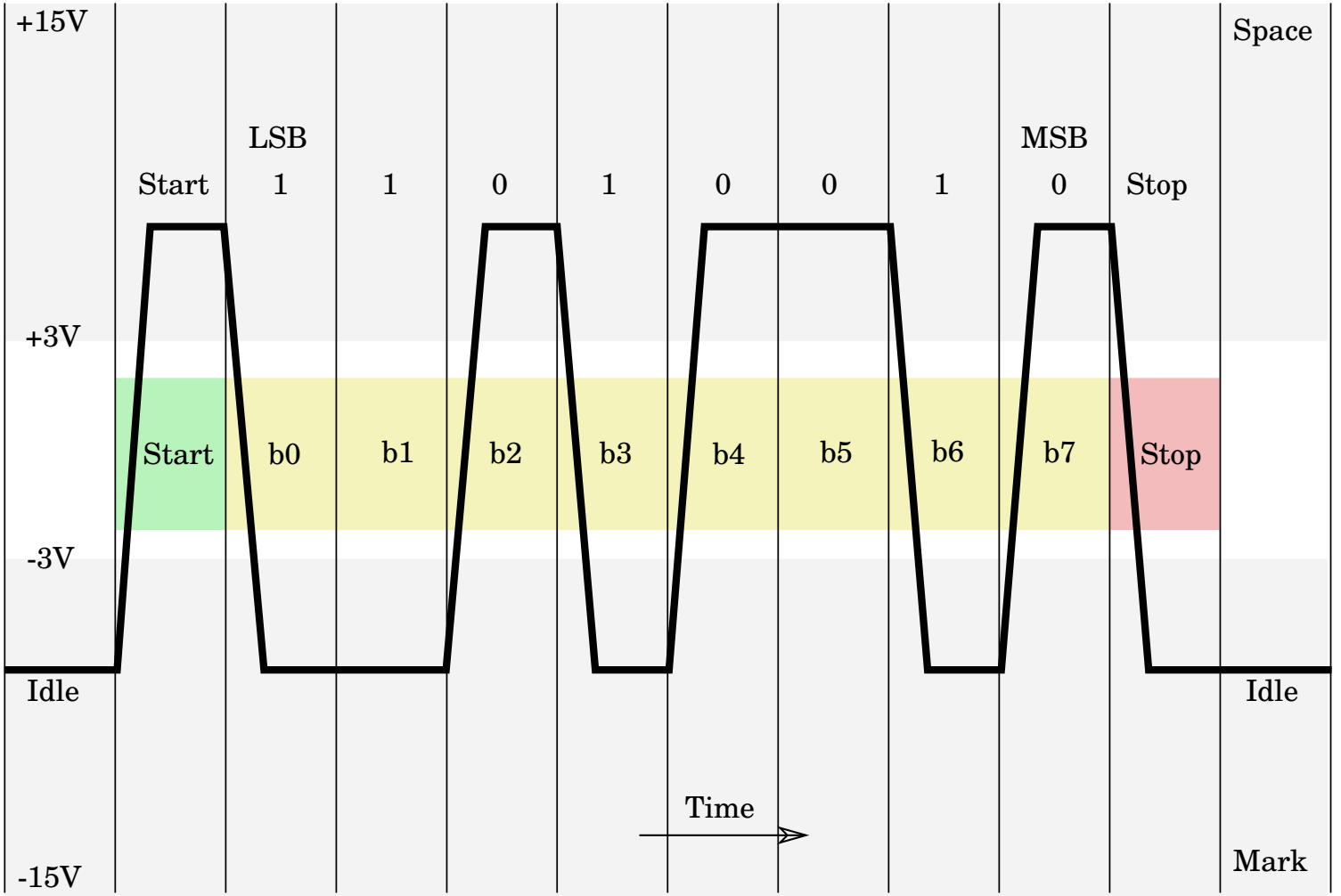
- Bus série asynchrone
- Topologies supportées:
  - Point → Point (en pair à pair)
- Full-duplex (ou simplex)
- Spécification électriques:
  - '0' logique codé par tension positive (3v < Vlow < 25v)
  - '1' logique codé par tension négative (-25v < Vhigh < -3v)
  - zone non définie (-3v < Vidle < 3v)





RS232, Exemple de trame

Transmission 8N1

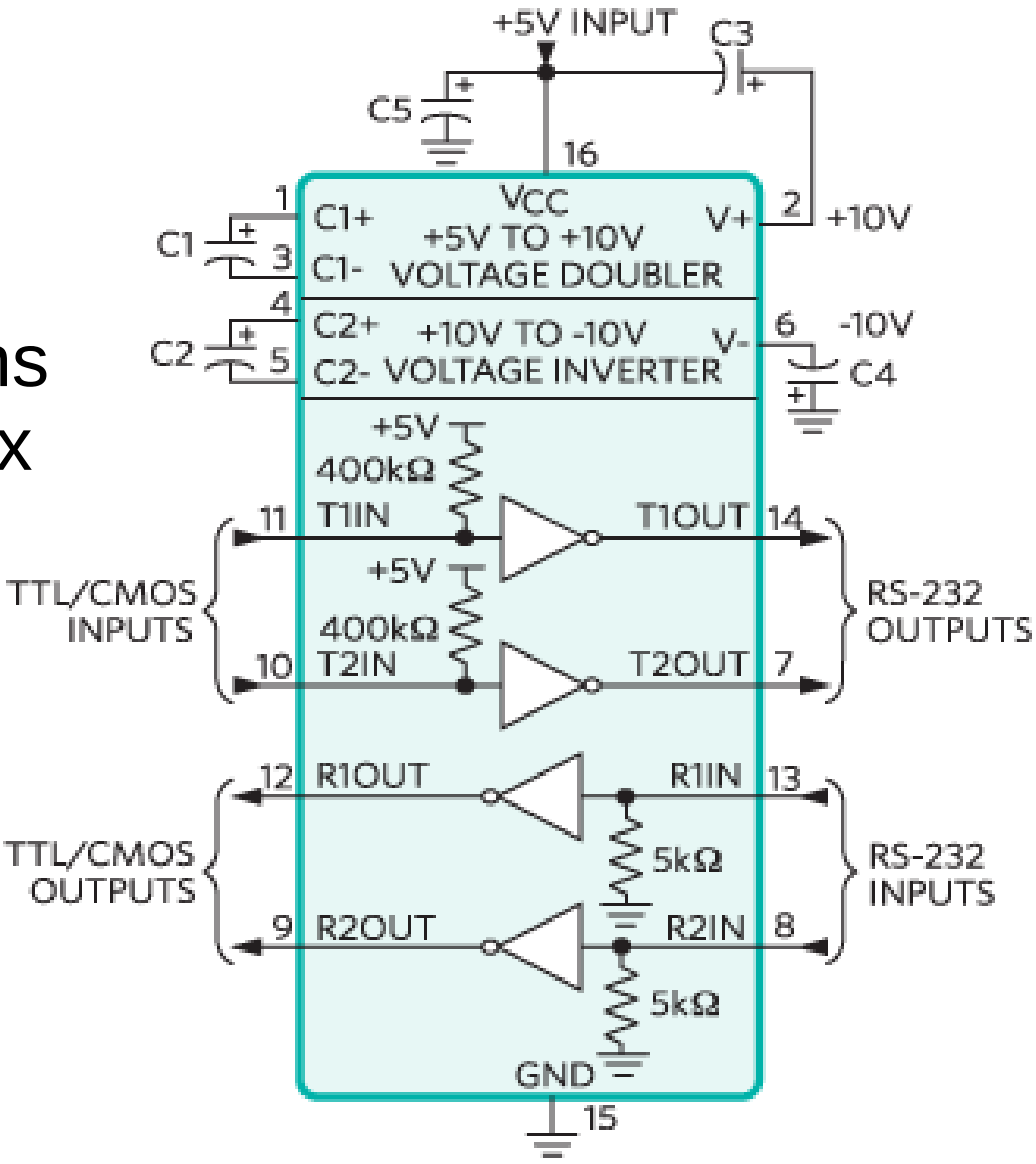


Source : Wikipedia

RS232, PHY

- Composant PHY
  - Ex MAX232:

    TTL → RS232
  - Génération des tensions utilisées par les signaux réalisée par le composant grâce à un circuit à pompe de charge



RS232, Câblage

- 3 conducteurs minimum: RX, TX, GND
  - Connecteurs DE-9 ou DB-25
- Autres conducteurs pour
  - Contrôle de flux
  - Détection de connexion
- Longueur:
  - 15m sur câble classique
  - Jusqu'à 300m sur câble faible capacité
- Vitesses:
  - 2400 bauds → 115200 bauds (4800, 9600, 38400, 57600, 115200)

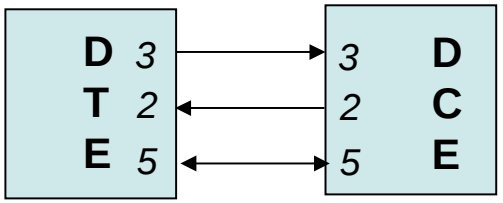
RS232, Exemple de cablage

2 types d'équipements :

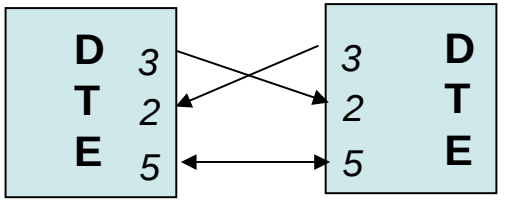
DTE *(Data Terminal Équipement) : Terminaux et PCs qui transmettent et reçoivent les données.*

DCE *(Data Communication Équipement) : Modems qui transfèrent les données.*

***Note**: Les définitions des Pins du RS232 sont souvent données par rapport à DTE*

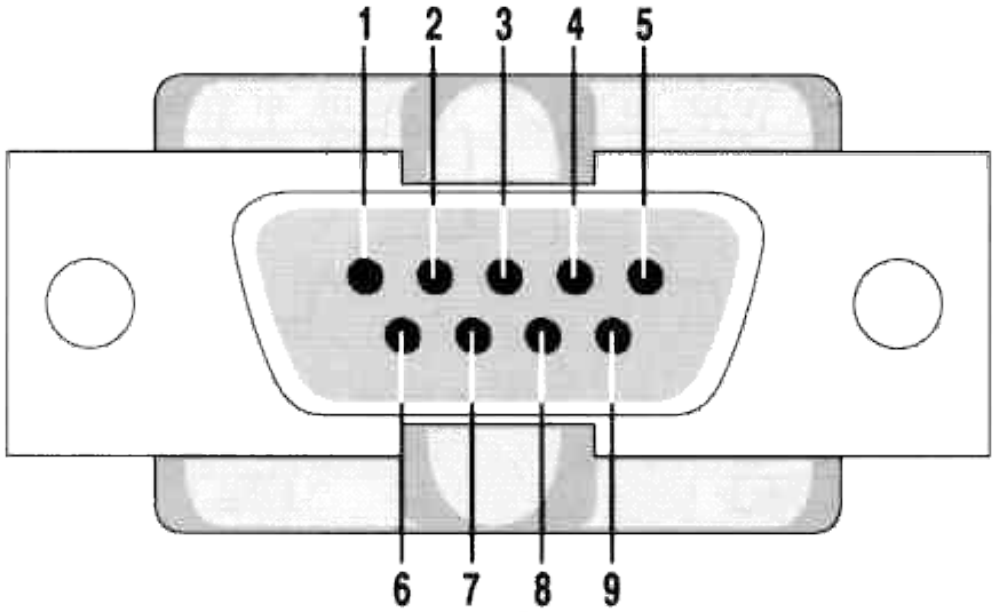


Connexion PC-Modem avec connecteurs DE9 par câble DROIT



Connexion PC-PC avec connecteurs DE9 par câble CROISE

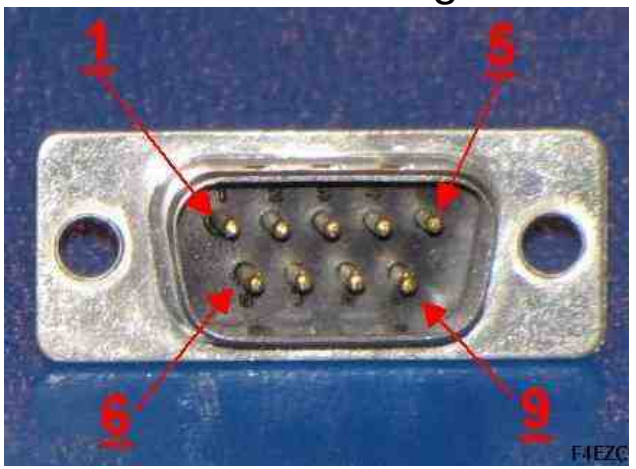
RS232 brochage DTE



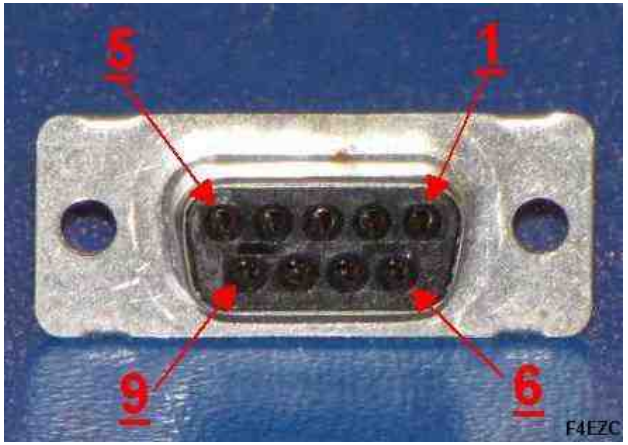
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Source: www.arcelect.com

Numérotation des broches différente selon le genre



Sur connecteur mâle



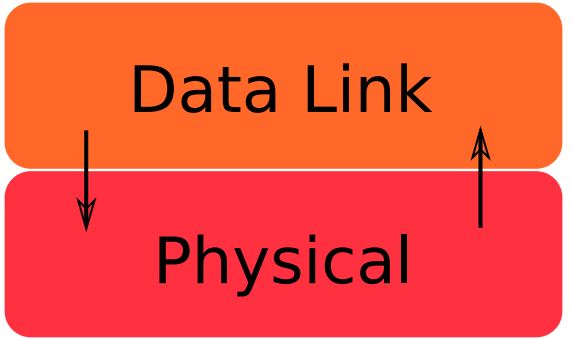
Sur connecteur femelle

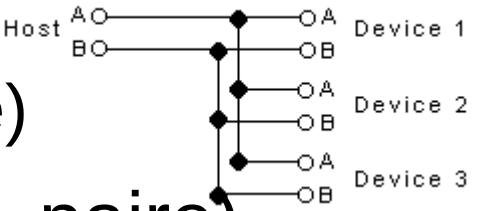
Source : http://radio.pagesperso-orange.fr/RS232.htm

RS422, Carte d'identité

- Bus série asynchrone
- Topologies supportées:
  - Point → Point (en pair à pair)
  - Point → Multi-dropped (10 max),

    1 seul émetteur (en maître-esclave)
- Full-duplex (2 paires), Half-duplex (1 paire)
- Spécifications électriques:
  - Transmission différentielle : +5v ou -5v entre les deux conducteurs
  - Valeur du bit encodée par le signe de la tension différentielle





RS422

- Câblage:
  - RX+, RX-, TX+, TX-
- Longueur:
  - Jusqu'à 1500m
- Vitesse
  - 100 kbauds → 10Mbauds

RS422, PHY

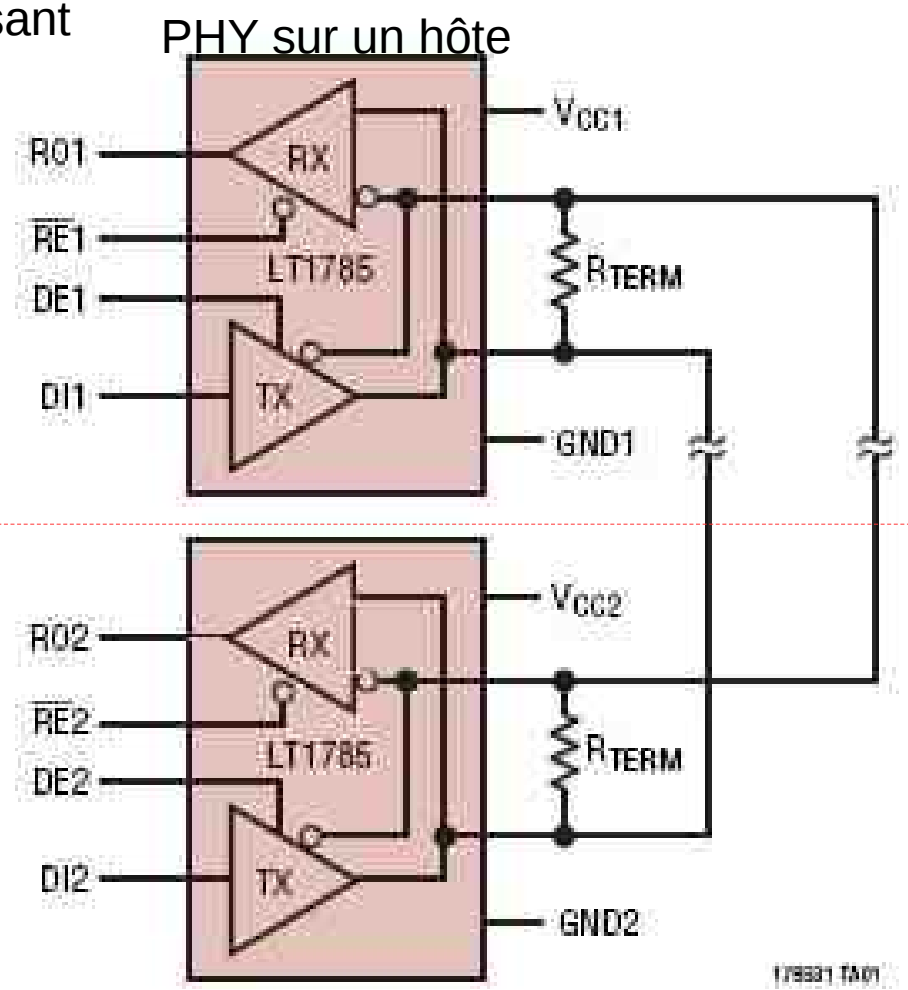
signaux au niveau du composant par exemple TTL

Signal reçu

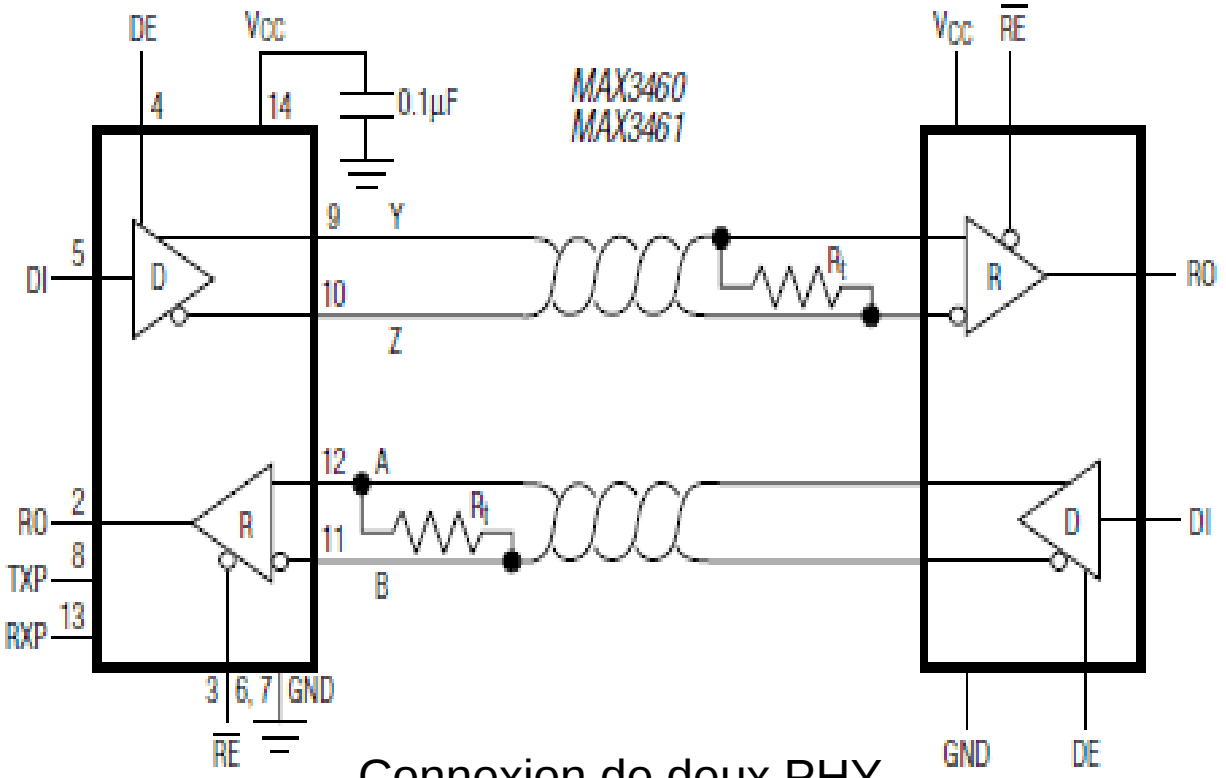
Autorisation de réception

Demande d'émission

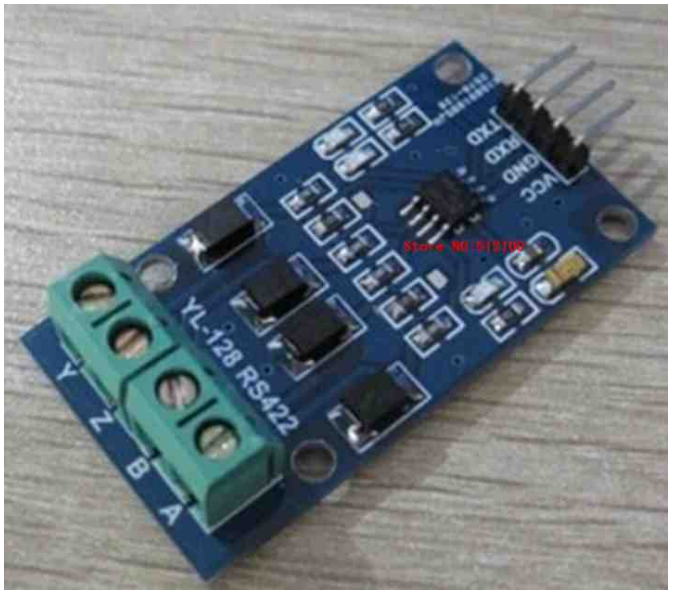
Signal à émettre



RS422, PHY



Connexion de deux PHY
à l'aide de deux paires torsadées
Utilisation de résistances de terminaison

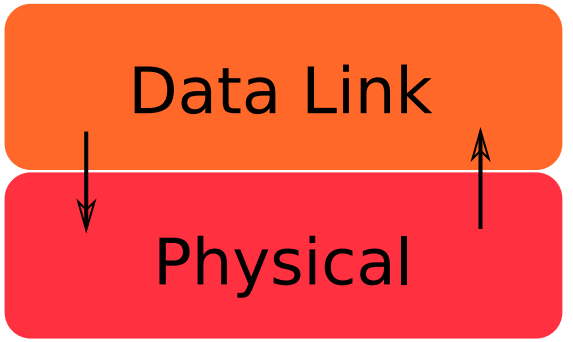


Module d'interface TTL<→RS422

RS485, Carte d'identité

- Bus série asynchrone
- Topologies supportées:
  - Point → Multi-points

    (en maître/esclaves)

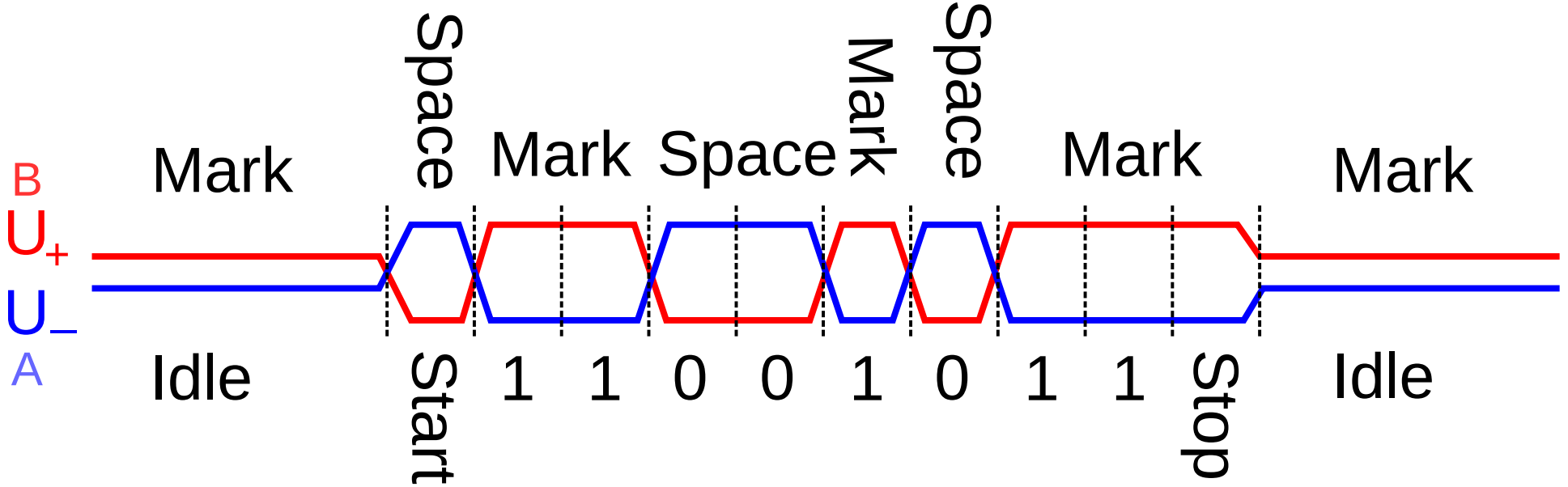


- Full-duplex (2 paires), Half-duplex (1 paire)
- Spécification électriques:
  - Transmission différentielle
  - +200mv, -200mv tension différentielle
  - Valeur du bit encodée par le signe de la tension différentielle

RS485

- Câblage:
  - A (inverting pin), B (non-inverting pin), C (ground)
- Longueur:
  - 10m → 1200m (selon vitesse)
- Vitesse
  - 100 kbauds → 35Mbauds

RS485, exemple de trame



Source : Wikipedia

UART et contrôle de flux

- Mécanisme permettant de s’assurer qu’un dispositif est prêt à recevoir une ou plusieurs données avant de lui envoyer
- Complémentaire à l’utilisation des files (FIFOS)
- Plusieurs réalisations possibles

Contrôle de flux **matériel**

- Spécifié uniquement sur RS-232 :Héritage du modem
- Procédure de la poignée de main : “Handshake”
- Utilisation de deux signaux supplémentaires **pour un sens de communication**
  - RTS : Request To Send
  - CTS : Clear To Send
- Topologie maître->esclave :
  - DTE : Data Terminal Equipment (Envoie des données)
  - DCE : Data Communication Equipment (Reçoit des données)

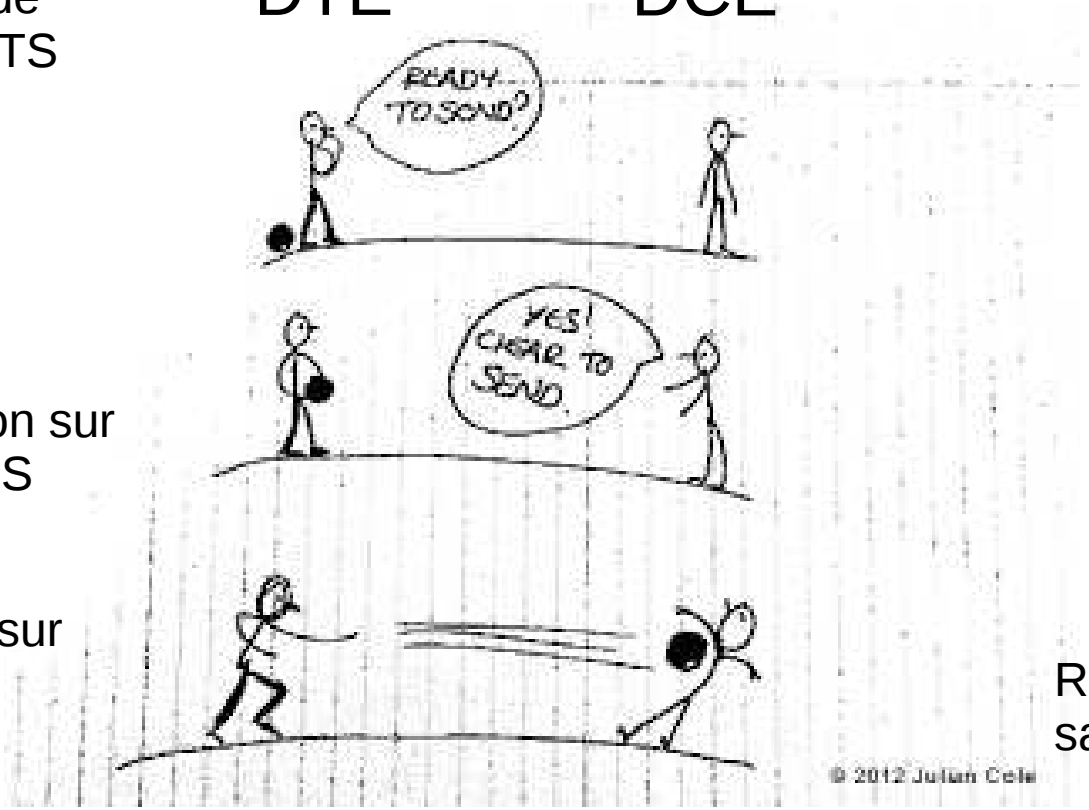
Contrôle de flux **matériel**

- Procédure RS232 RTS/CTS :

Le DTE souhaite transmettre des données au DCE

DTE DCE

Émet la demande sur sa broche RTS (sortie)



Reçoit la demande sur sa broche CTS

Émet l'autorisation sur sa broche RTS

Reçoit l'autorisation sur sa broche CTS (entrée)

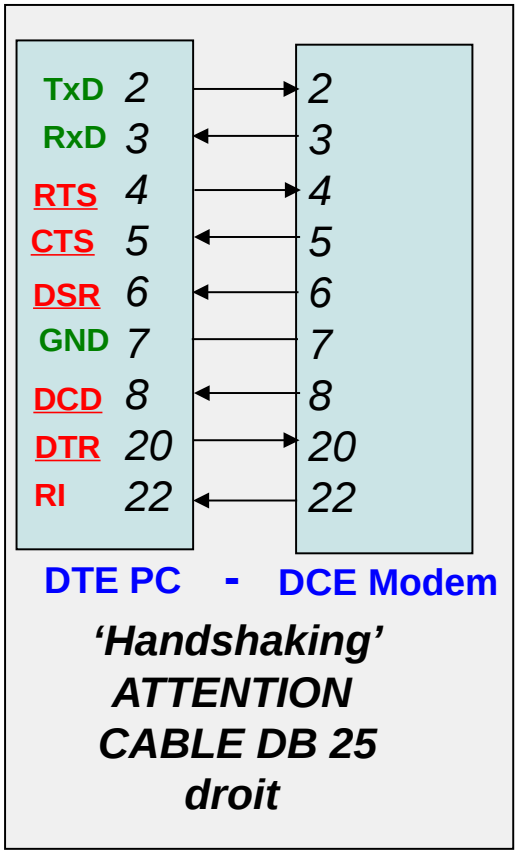
Émet les données sur sa broche TX

Reçoit les données sur sa broche RX

Le nom des signaux est significatif du coté de l'émetteur des données (DTE)

Signaux d'échange du RS232

Pour assurer une transmission fiable entre 2 systèmes, le transfert de données doit être coordonné → c'est le rôle des signaux de Handshaking.



DTR (Data Terminal Ready): Le Terminal (PC port COM) est prêt après sa mise en marche. S'il y a problème avec le port COM, ce signal ne peut être activé.

DSR (Data Set Ready): Le DCE (Modem) est prêt après sa mise en marche. S'il y a un problème avec le la connexion téléphonique, ce signal ne peut être activé.

RTS (Request To Send): Quand le DTE (PC) a un octet à transmettre, il active RTS pour le signaler au modem.

CTS (Clear To Send): En réponse au RTS, le modem envoie CTS pour signaler au PC qu'il est prêt à recevoir la donnée maintenant. Et le PC commence la transmission.

DCD (Data Carrier Detect): Le Modem indique, via DCD, au PC que le contact entre lui et un autre modem est établi pour recevoir la donnée que la PC lui a envoyé.

RI (Ring Indicator): Le Modem indique au PC que le téléphone sonne.

Possibilité d'utiliser DTR/DSR comme signaux de contrôle de flux pour la communication dans le sens DCE vers DTE

Contrôle de flux **Logiciel**

- Utilisation d'une solution logicielle : Utilisation de caractères spéciaux transmis sur le même support physique que les données
  - XON (11h) et XOFF (13h)…
  - Latence induite par la sérialisation/désérialisation
- Ne nécessite pas de signaux supplémentaires
- Héritage des imprimantes texte
- Contrôle de flux asymétrique

Contrôle de flux **Logiciel**

- Procédure
  - Le receveur des données envoie une autorisation d’envoi : XON
  - L’émetteur débute l'envoi des données
  - Le receveur envoie XOFF dès qu'il ne peut plus recevoir de données (par exemple parce que sa FIFO de réception va bientôt déborder)
  - L’émetteur stoppe l'envoi des données
  - Le receveur envoie XON dès qu'il peut de nouveau recevoir des données (par exemple parce que sa FIFO de réception a de nouveau suffisamment de place)
- Dans certains cas, si des données correspondant aux caractères spéciaux (XON/XOFF) doivent être envoyées, on utilise le caractère de transparence pour les préfixer, et on applique une opération logique à la donnée.

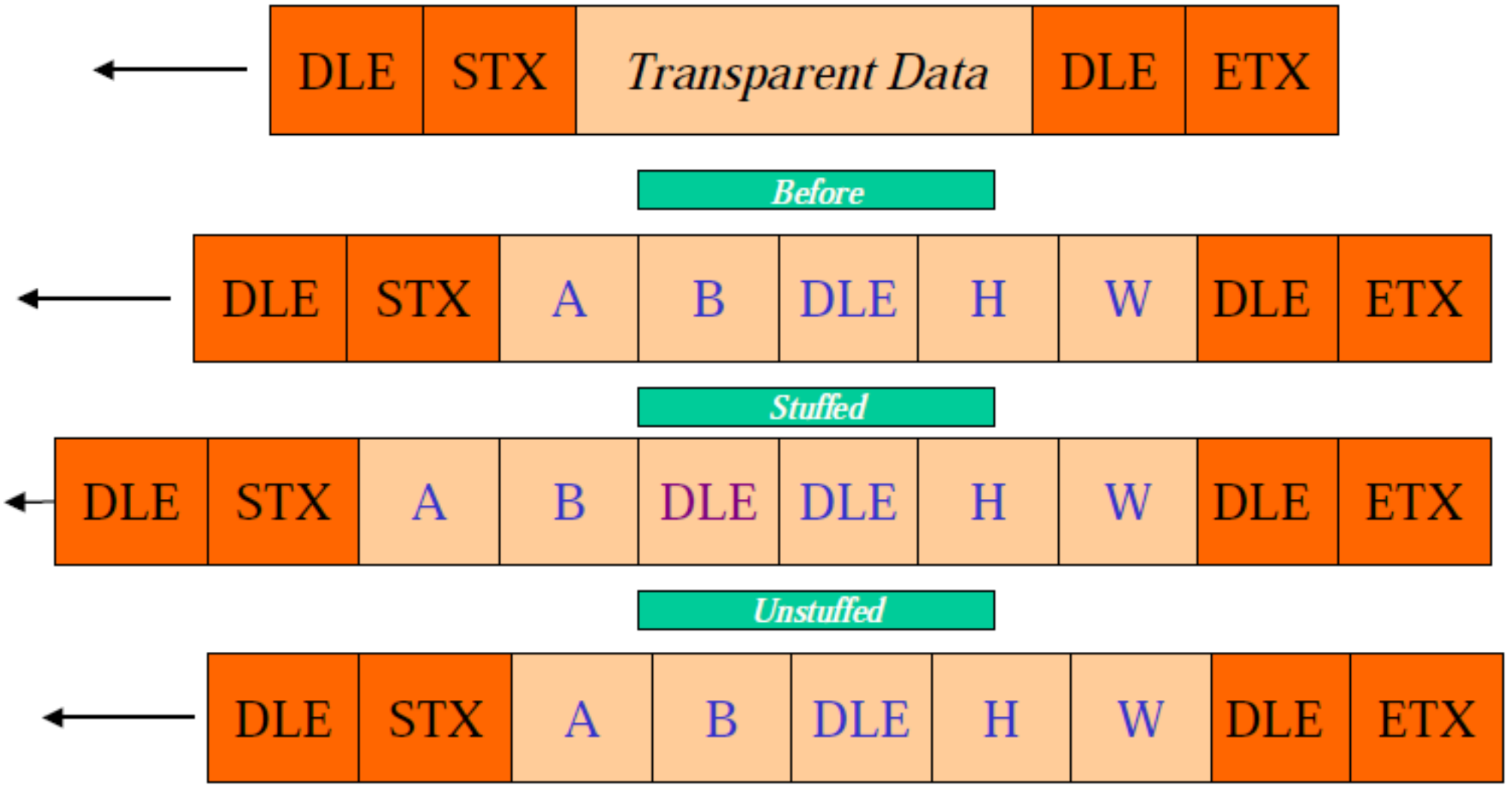
UART et couche réseaux

- Communication par paquets de données
  - Les messages ne sont généralement pas des octets isolés
- Ajout de caractères spéciaux au vocabulaire de communication
- Besoin d'un caractère de transparence pour signaler les caractères spéciaux

Protocole STX/ETX

- Protocole de transmission sur lien série STX/ETX
  - Niveau 2 du modèle OSI (découpage des données)
  - STX (02h) début de paquet,
  - ETX (03h) fin de paquet,
  - DLE (010h) Fin du canal de données (caractère d'échappement) :
    - Indique que l'octet à suivre est à interpréter comme un caractère de contrôle
    - Si la donnée à transmettre contient DLE, DLE est ajouté devant la donnée

Protocole STX/ETX

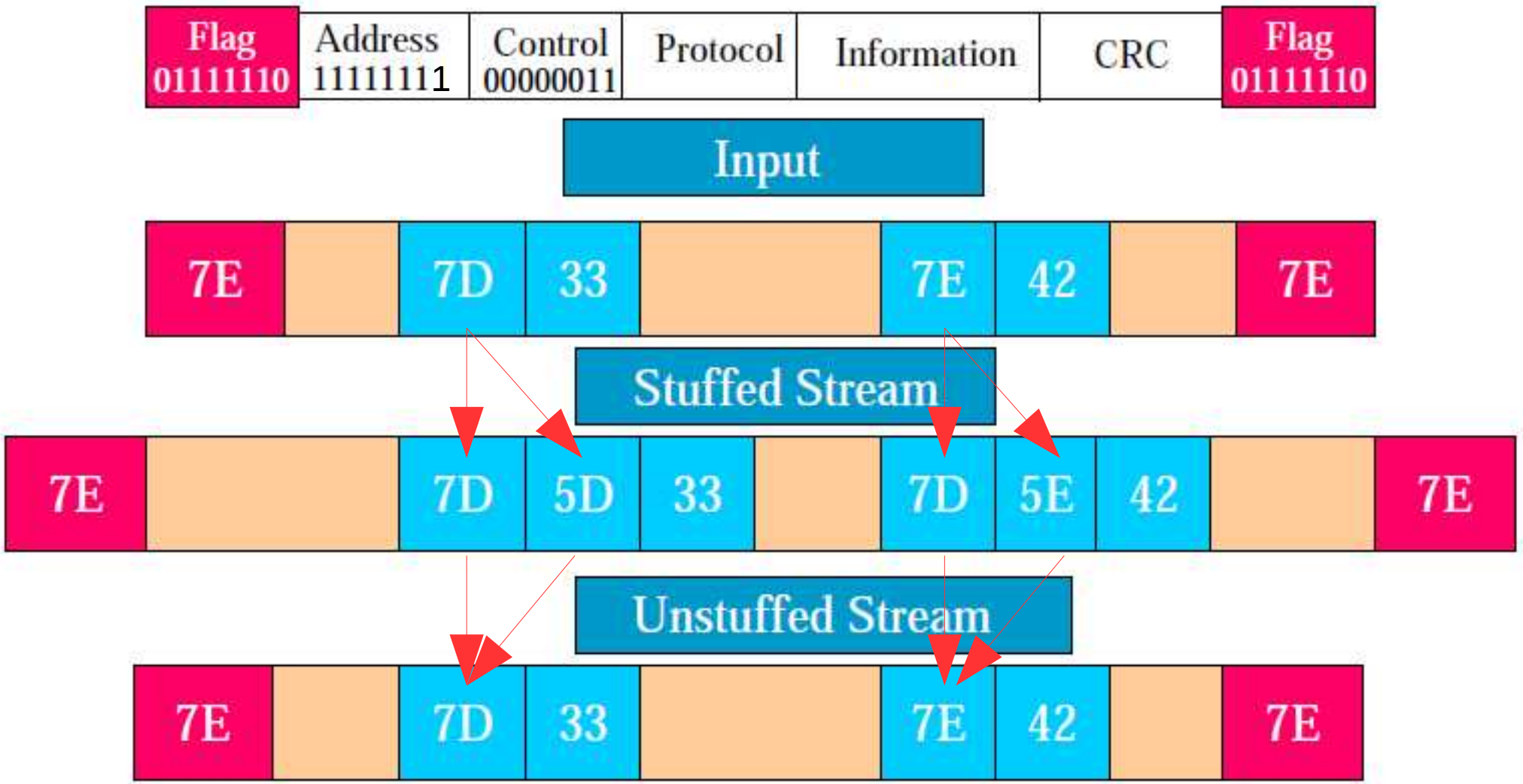


Protocole HDLC

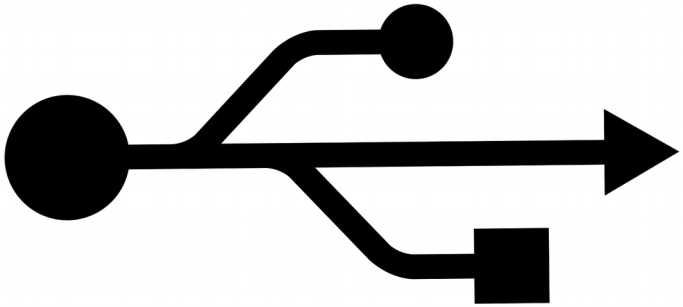
- Protocole de transmission sur lien série HDLC : (High-Level Data Link Control)
  - Niveau 2 du modèle OSI :
    - découpage des données
    - Adressage
    - Contrôle d’erreur par CRC
  - Utilisation d'un “fanion” (flag) pour la synchronisation (délimiteur):07Eh.
    - Les trames HDLC peuvent être envoyées les unes derrière les autres : dans ce cas, le fanion de fin de la première trame peut être mis en commun et servir de fanion de début pour la trame suivante
  - Utilisation d'un caractère d'échappement : 07Dh
    - Bourrage d'octet (byte-stuffing) pour la transparence :Les données dont les codes sont 07Eh ou 07Dh sont masquées en préfixant par le caractère d'échappement et en inversant leur bit 5.

https://fr.wikipedia.org/wiki/High-Level_Data_Link_Control

Protocole HDLC



Bus USB (<=2.0)



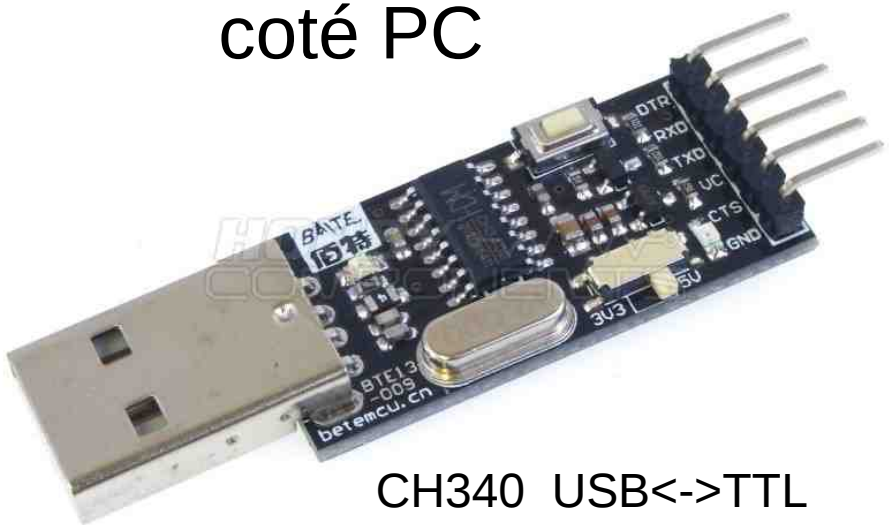
- Carte d'identité:
  - Bus série asynchrone
  - Topologies supportées:
    - Point → Multi-points grâce à un (des) concentrateur(s) (HUB)
      - 1 concentrateur racine dans l'hôte puis jusqu'à 5 concentrateurs en cascades
      - Jusqu'à 127 appareils sur un même bus
    - maître(host) → esclaves (device), éventuellement négociable à la volée (On The Go)
  - Half-duplex (1 seule paire différentielle pour USB<=2.0)
  - Débits
    - 1,5Mbits/s en USB 1.0 Low speed
    - 12Mbits/s en USB 1.1 High speed
    - 480Mbits/s en USB 2.0
    - 4Gbits/s en USB 3.0
    - 10Gbits/s en USB 3.1
  - Spécification électrique:
    - Transmission différentielle
    - 0→ 3.6v tension différentielle
    - 'J' : 2.8v → 3.6v en basse vitesse (USB1.0)
    - 'K' : 0v → 0.3v en basse vitesse (USB1.0)
    - '0' codé par une transition d'état, '1' codé par aucune transition : 00010b codée en JKJKKJ ou KJKJJK
    - Alimentation du périphérique (device) par le bus
      - 5v 500ma jusqu'à USB2.0
      - Tensions et courants plus élevés à partir de USB3.0

Bus USB

- Permet l'adressage dans le périphérique (adressage logique)
  - Endpoints (exemple : clavier + trackpad)
- Définit plusieurs types de couches logiques
  - Interrupt, Isochrone, Bulk
- Définit des classes de périphériques:
  - HID, HUB, Video, Audio ...
- Nécessité d'acheter un identifiant unique auprès du USB Implementers Forum
  - Vendor ID+Product ID

Composants de conversion USB

- FTDI, Silicon-Labs, Cypress
  - Convertit le standard USB vers: UART, I2C, SPI
    - bit-banging via IOs
    - FIFO matérielles (pour débits et duplex différents)
  - Pas besoin d'acquérir un USB vendor-id
  - Dans le cas USB→ UART, utilisation simplissime coté PC

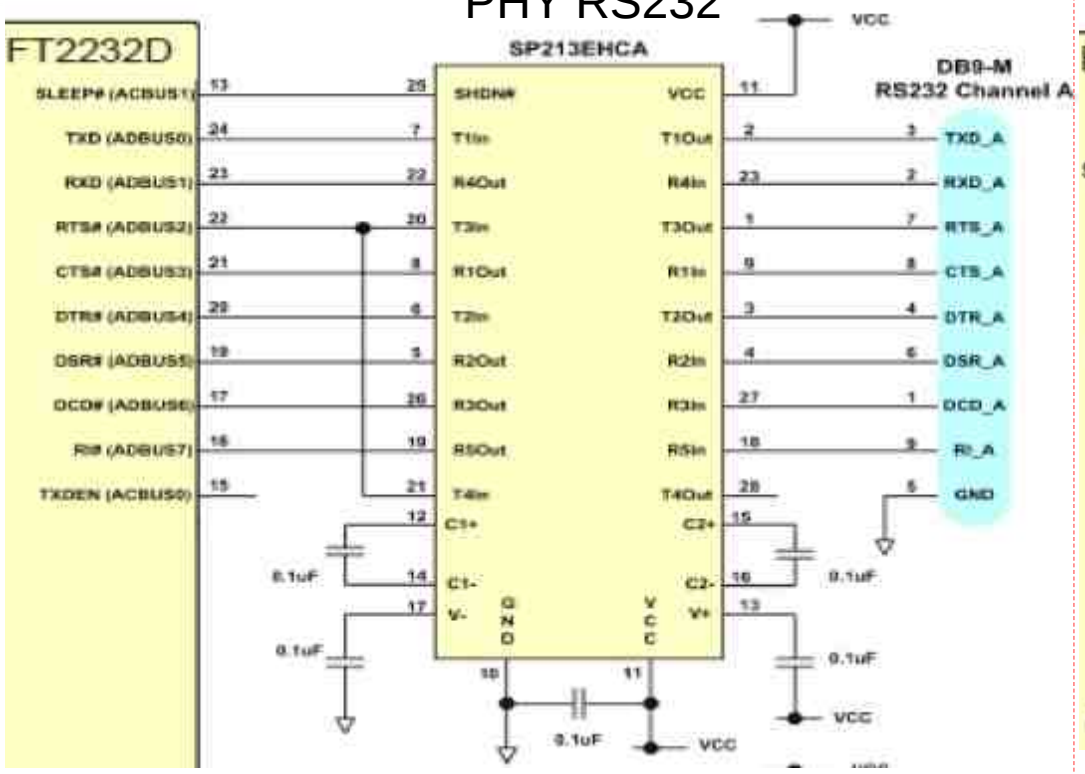


CH340 USB<->TTL

FTDI2232H USB2.0 HS USB<->TTL

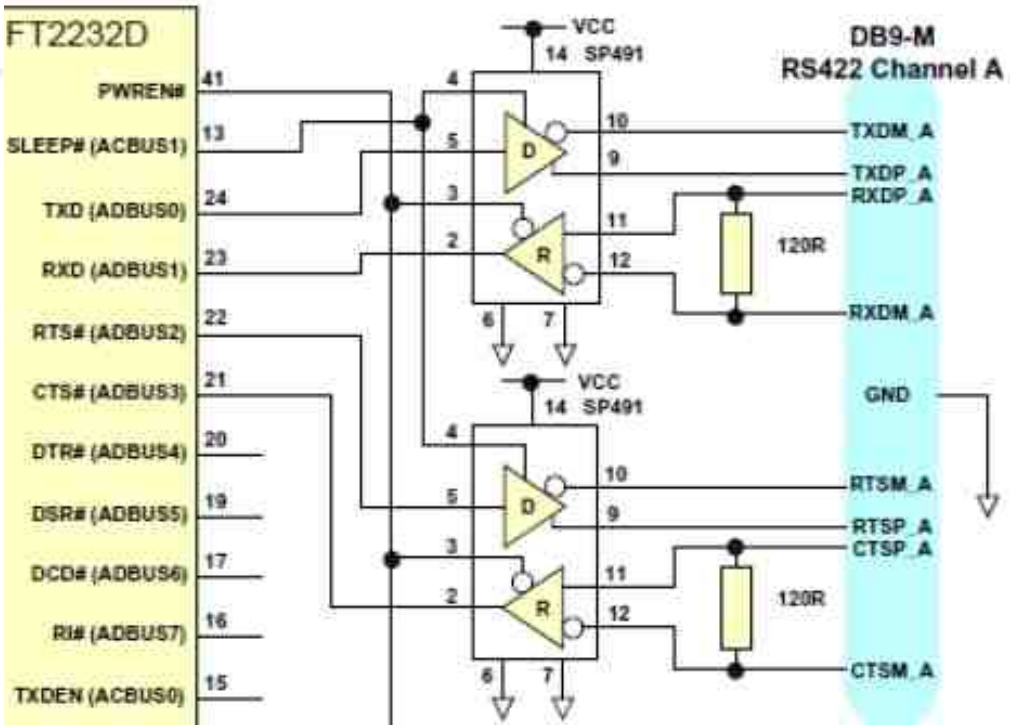
Composants de conversion USB

PHY RS232



Câblage en convertisseur USB<->RS232

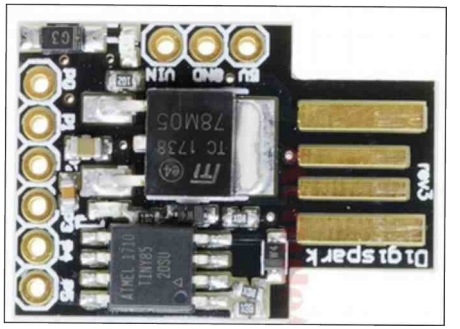
PHY RS422



Câblage en convertisseur USB<->RS422

USB sur microcontrôleur

- USB device émulé (V-USB pour Atmel):
  - Limité à USB1.0



Carte microcontrôleur Digispark

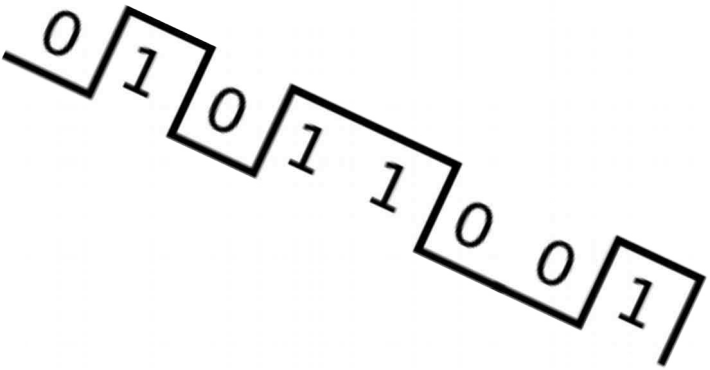
- USB device/host intégré (TI, Atmel, Freescale, Microchip ...):
  - Jusqu'à USB2.0
  - Intègre un PHY USB
  - Intègre une Stack USB logicielle
    - fournie par le fabricant
    - Alternatives open-source
  - Vendor-ID fournit par le fabricant pour le prototypage

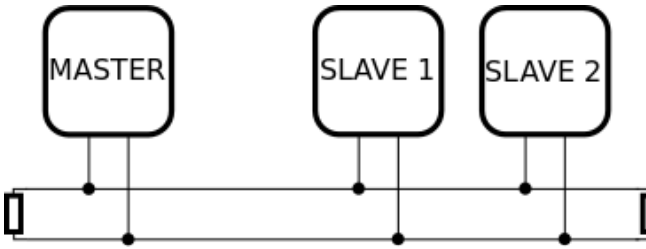
Descripteur USB

- Configuration de l'interface
- Configuration du descripteur
- Installation des “Endpoints”
- Installation des Call-back pour les endpoints

```
const USB_Descriptor_Device_t PROGMEM DeviceDescriptor =
{
    .Header                 = {.Size = sizeof(USB_Descriptor_Device_t), .Type = DTYPE_Device},

    .USBSpecification       = VERSION_BCD(01.10),
    .Class                  = USB_CSCP_NoDeviceClass,
    .SubClass               = USB_CSCP_NoDeviceSubclass,
    .Protocol               = USB_CSCP_NoDeviceProtocol,

    .Endpoint0Size          = FIXED_CONTROL_ENDPOINT_SIZE,

    .VendorID               = 0x03EB,
    .ProductID              = 0x2042,
    .ReleaseNumber          = VERSION_BCD(00.01),

    .ManufacturerStrIndex   = STRING_ID_Manufacturer,
    .ProductStrIndex        = STRING_ID_Product,
    .SerialNumStrIndex      = NO_DESCRIPTOR,

    .NumberOfConfigurations = FIXED_NUM_CONFIGURATIONS
};
```

Bus Série synchrone:
Bus SPI
Bus I2C



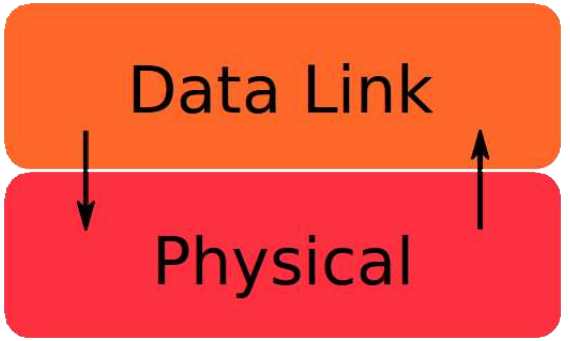


Bus Série Synchrone

- Le bus transmet les symboles (bit) synchrones avec une horloge elle aussi disponible sur le bus.
- Avantages:
  - Vitesse du bus (pas de reconstitution de la synchronisation)
- Inconvénients:
  - Câblage (nombre de conducteurs et distance)
  - Sensible au bruit sur le signal d’horloge

Bus SPI

- SPI : “Serial Peripheral Interface”
- Cartes d'identité:
  - Bus série synchrone
  - Topologies supportées:
    - Point → multi-points en maître-esclave
    - Daisy-chain
  - Full-duplex (ou éventuellement simplex)

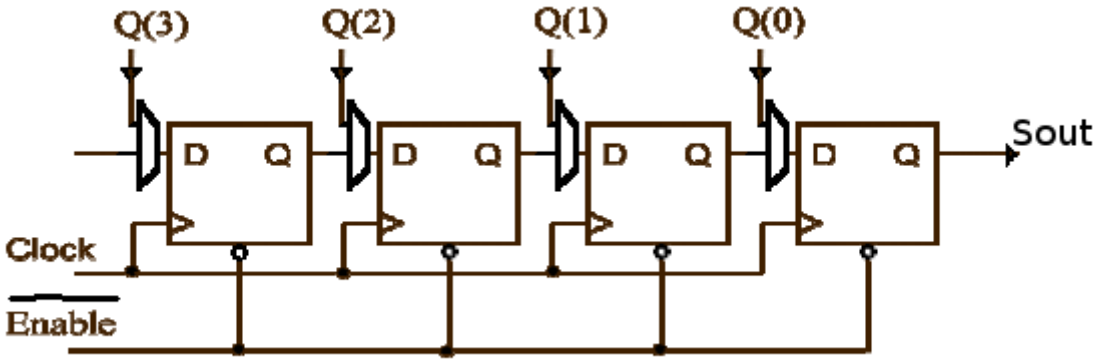


Bus SPI, Spécifications

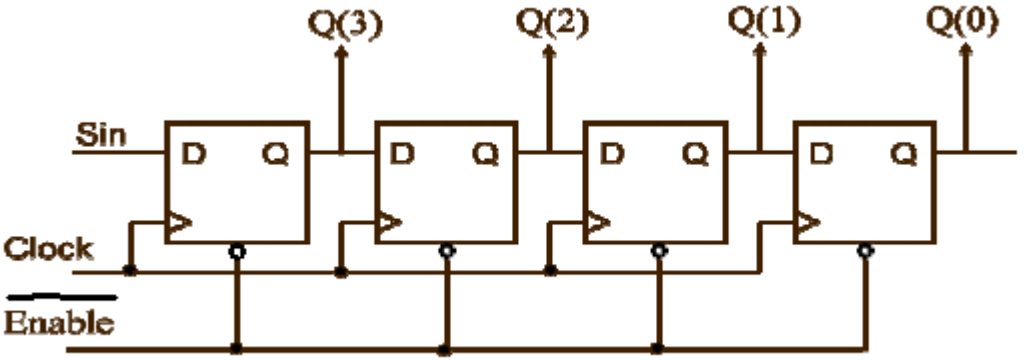
- Spécification électrique :
  - Niveaux TTL (5v), LVTTL (3.3v)...
- Câblage:
  - 4 + N conducteurs (GND, MISO, MOSI, SCK, 1 SS par esclave)
- Vitesses :
  - Pas de spécification de vitesse
  - En pratique : 100kHz → 64MHz
- Longueur de lignes:
  - Principalement utilisé comme bus de carte
  - Longueur centimétrique

Bus SPI, Implémentation matérielle

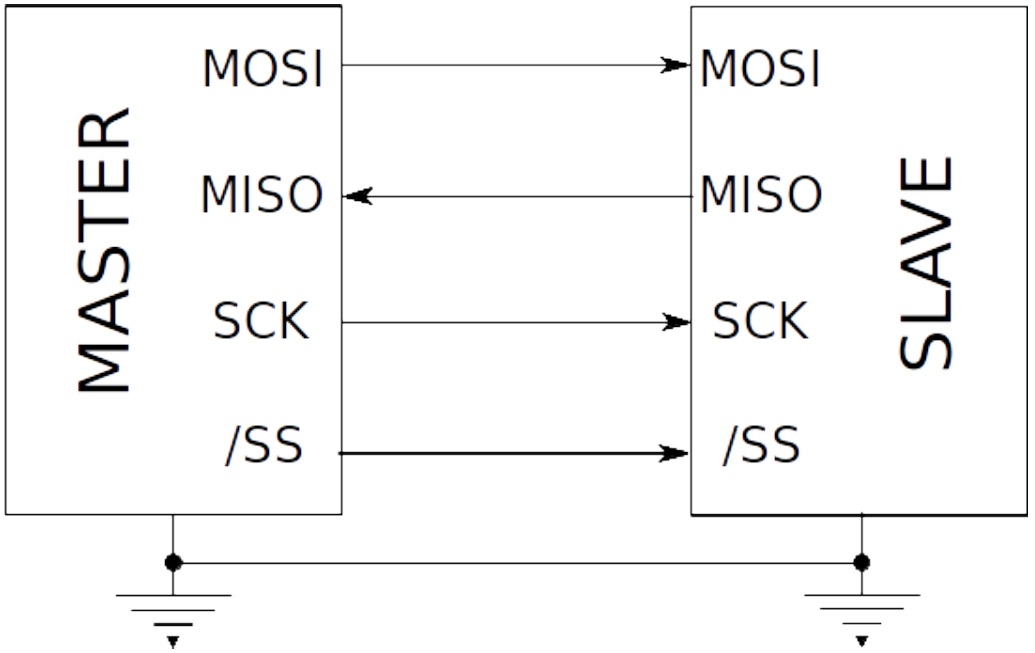
- Registre à décalage entrée parallèle/sortie série en émission :



- Registre à décalage entrée série/sortie parallèle en réception (/Enable du registre sur /SS actif à l'état bas) :

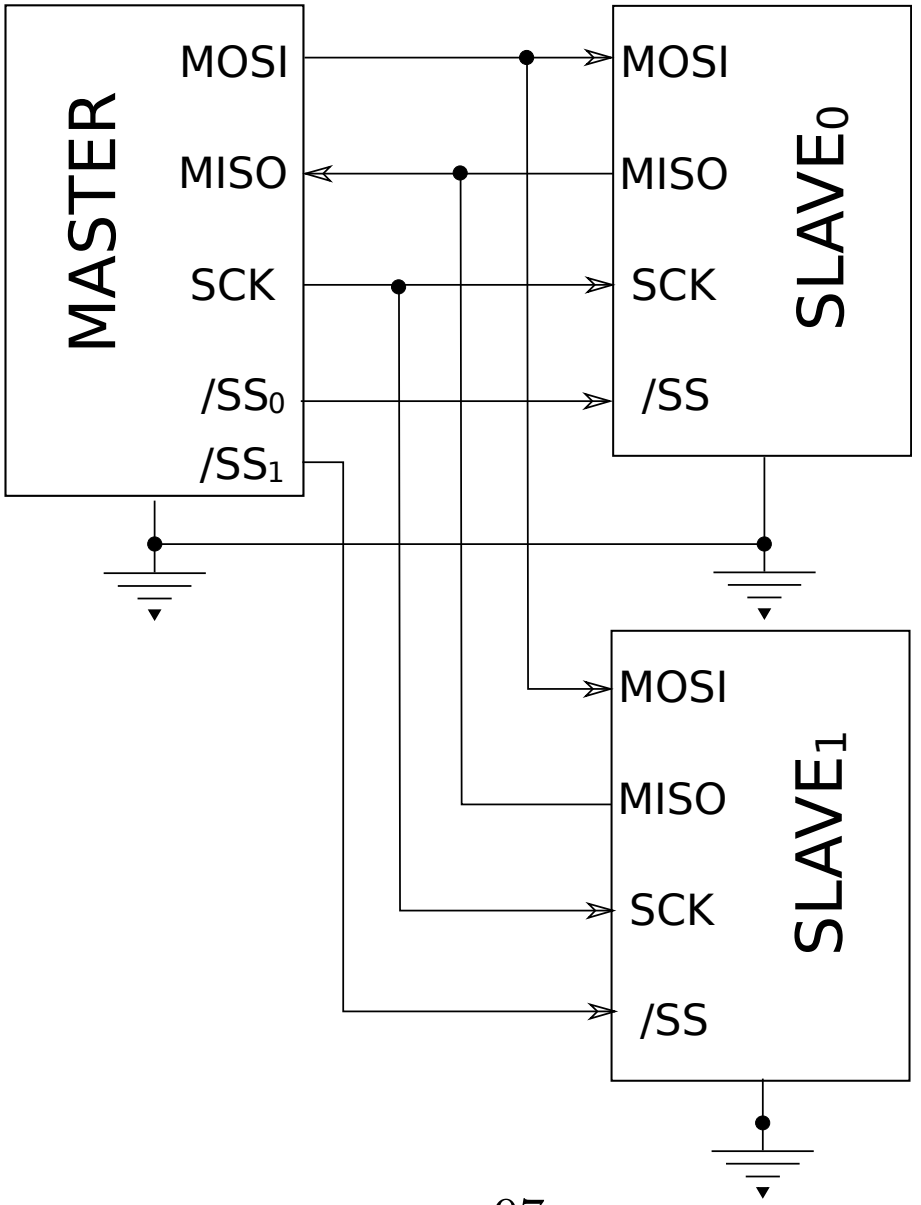


Bus SPI, Connexion

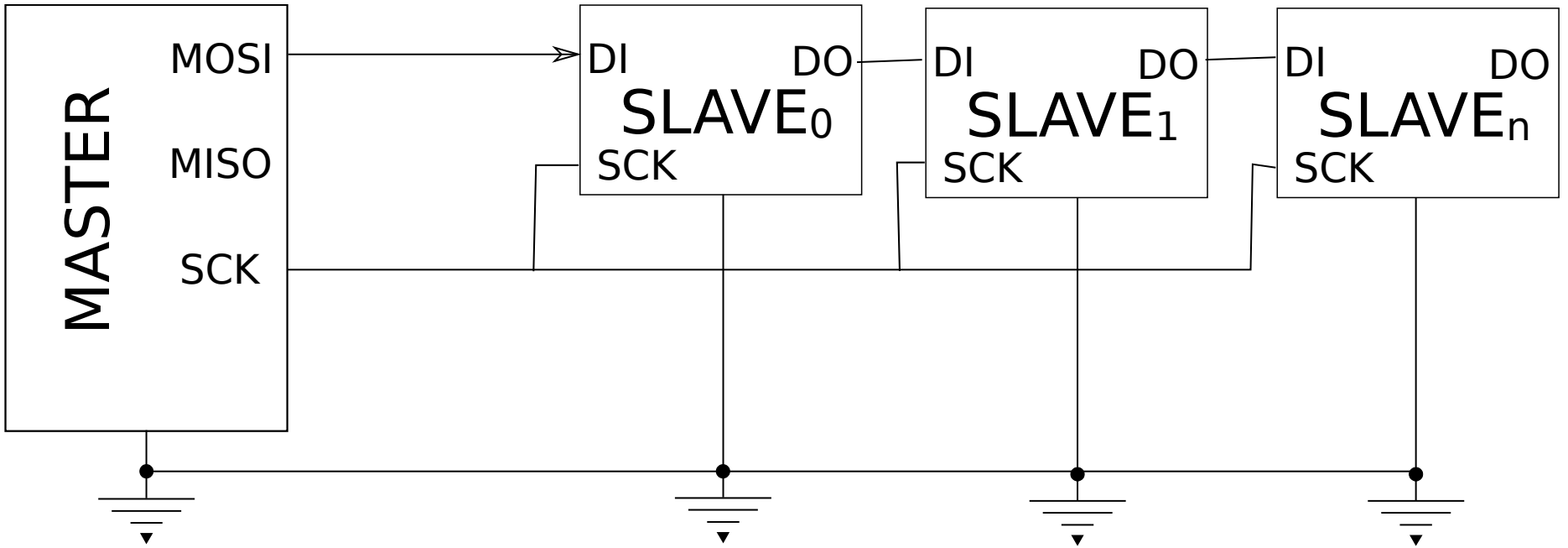


MOSI : Master Out Slave In, synonymes : DO (Data Out) coté maître DI (Data In) côté esclave
MISO : Master In Slave Out, synonymes : DO (Data Out) coté esclave DI (Data In) côté maître
SCK : Serial ClocK, horloge de transmission
/SS : Slave Select, synonymes : /CS (Chip Select)

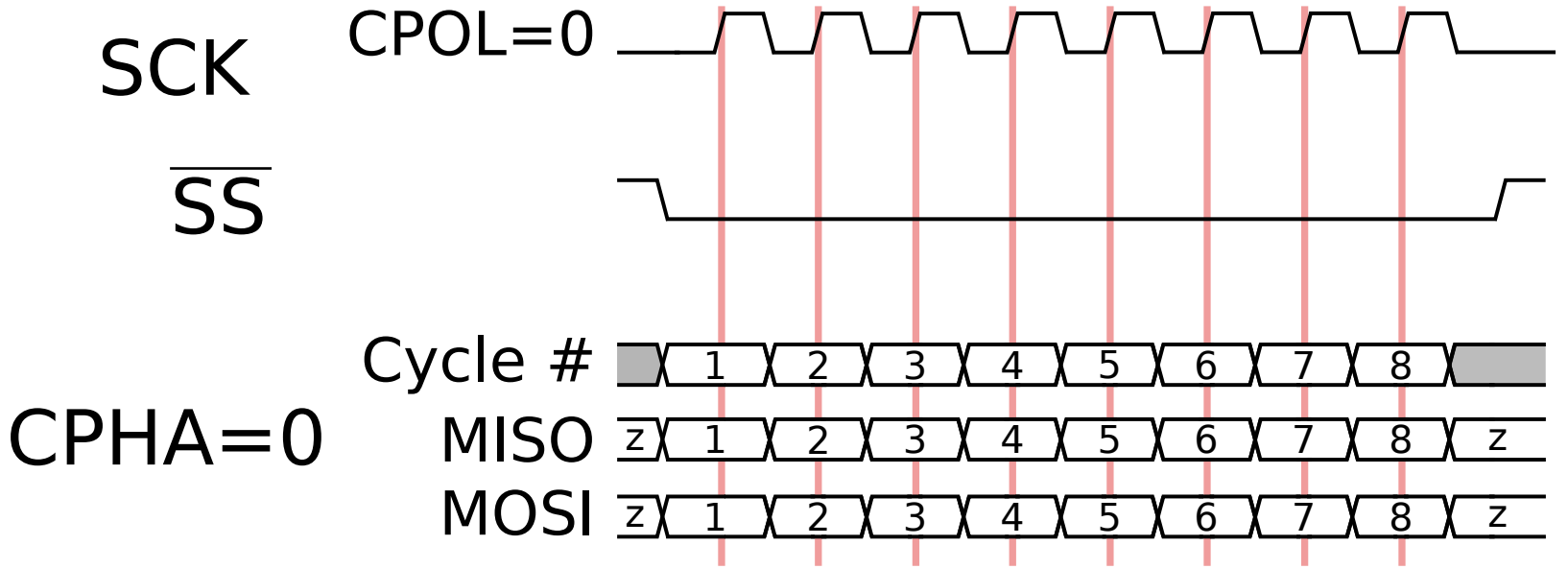
Bus SPI en topologie Étoile



Bus SPI en topologie Daisy Chain

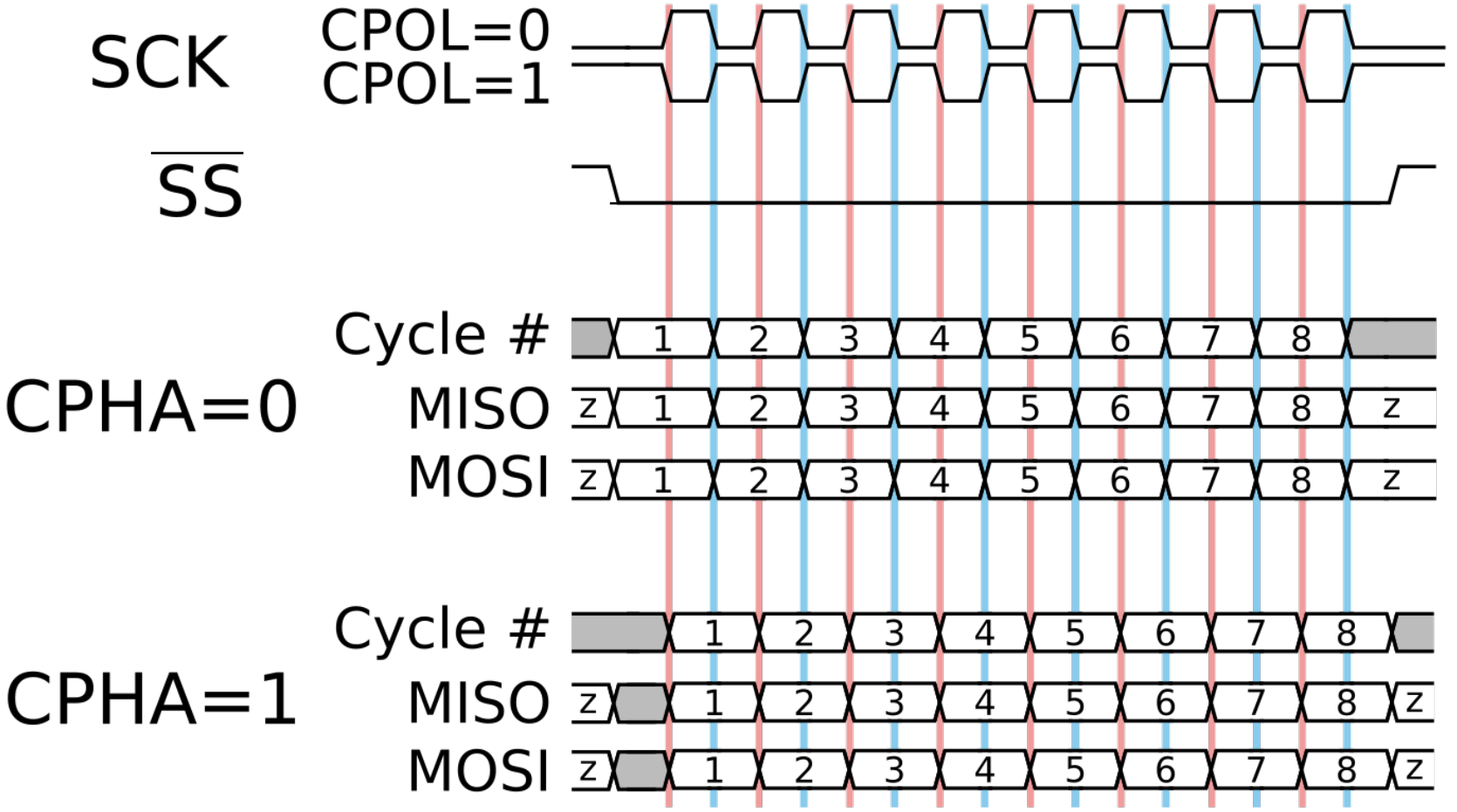


Bus SPI:Chronogrammes



Bus SPI:Modes

4 modes différents définis par la combinaison des paramètres CPOL et CPHA



Les composants d'un bus doivent utiliser le même mode pour échanger des données

Bus SPI

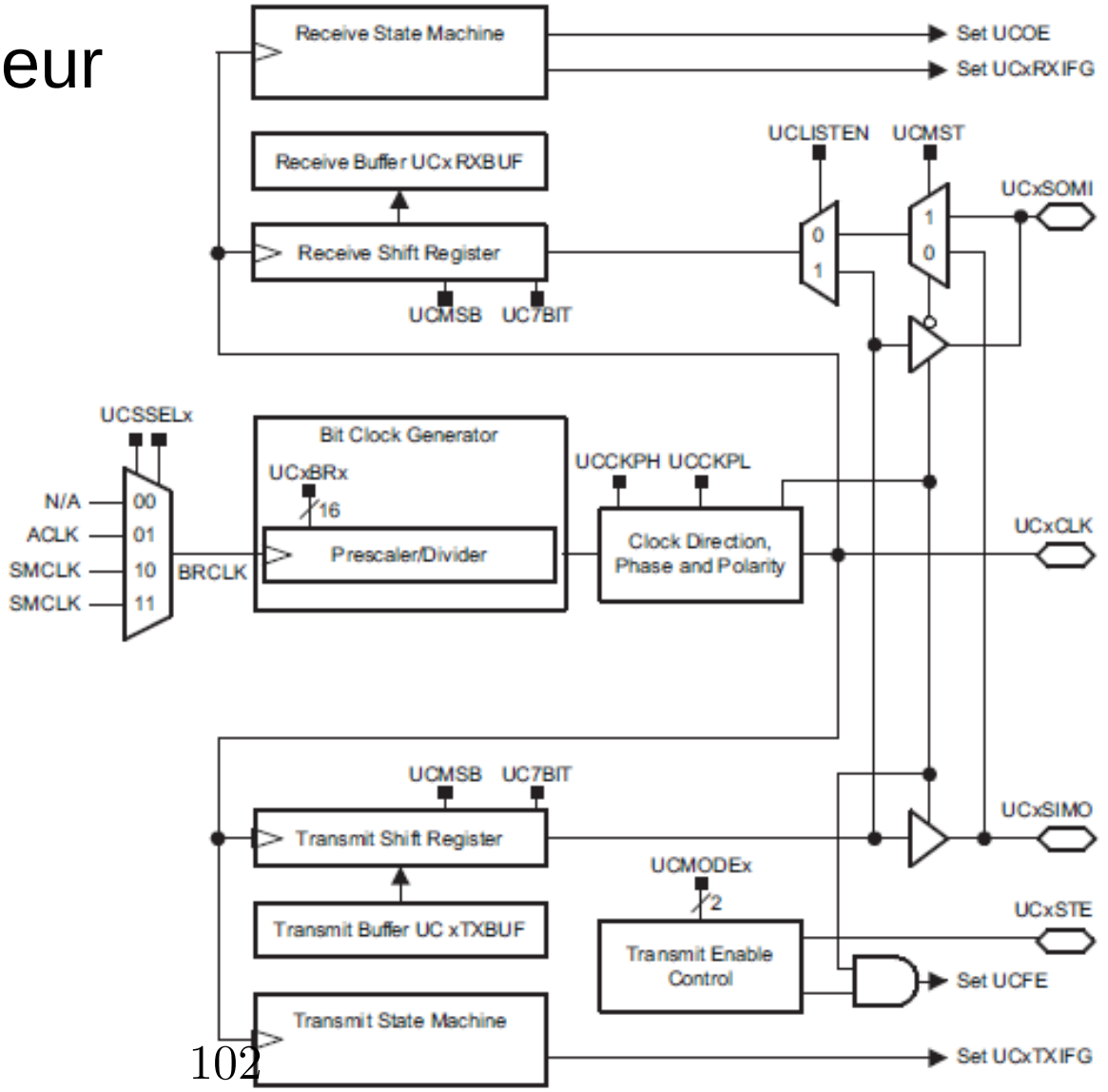
- Configuration:
  - Fréquence d’horloge
  - Mode SPI :
    - Phase de l'horloge (échantillonnage sur front montant/descendant)
    - État de l'horloge au repos (Polarité)

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

  - Configuration du/des Slave Select (GPIO ou broches dédiées)

Bus SPI

- Configuration périphérique intégré :
  - Ex: microcontrôleur Texas MSP430



Bus SPI

```
Void open_spi(){
	P1DIR |= ( 1 << (slave -> csPin)); // CS
	P1OUT |= ( 1 << (slave -> csPin)); // CS
	P1SEL |= BIT7 | BIT5 | BIT6 ; //SIMO & SOMI & SMCLK
	P1SEL2 |= BIT7 | BIT5 | BIT7 ; //select pin function
	UCB0CTL1 |= UCSWRST ; // reset configuration
	UCB0CTL0 |= UCCKPL | UCMSB | UCMST | UCSYNC ; //polarity, MSB first, master,
sync
	UCB0CTL1 |= UCSSEL_2 ; // clock source
	UCB0BR0 |= 64; //prescale by 64
	UCB0BR1     |= 0 ; // prescaler = BR0 + BR1 * 256
	UCB0CTL1 &= ~UCSWRST ; // config done
}
```

```
 unsigned char transfer_byte_spi(unsigned char val){
		UCB0TXBUF = val ;
		while(UCB0STAT & UCBUSY);
		return UCB0RXBUF;
 }
```

Bus SPI

```
void transfer_data_spi(int slave, unsigned char * send_buffer, unsigned char *
receive_buffer , unsigned char length){
	unsigned int i ;
	spiClearCs(slave);
	for(i = 0 ; i < length ; i ++){
		receive_buffer[i] = transfer_byte_spi(send_buffer[i]);
	}
	spiSetCs(slave);
}
```

Bus SPI

- Configuration périphérique émulé :

```
Void open_spi(){
        set_ss_output();
        set_sck_output();
        set_mosi_output();
        set_miso_input();
}
```

Bus SPI

```
unsigned char transfer_byte_spi(unsigned char val){
        unsigned char i = 0 ;
        unsigned char valBuf = val ; //valeur à transmettre
        unsigned char inBuf = 0 ;     //valeur reçue
        __delay_cycles(SPI_BB_CYCLE);
        for(i = 0 ; i < 8 ; i ++){ //MODE3
                if(valBuf & 0x80){
                        clr_sclk();
                        set_mosi();
                }else{
                        clr_clk();
                        clr_mosi();
                }
                valBuf = (valBuf << 1);
                __delay_cycles(SPI_BB_CYCLE);
                inBuf = (inBuf << 1) ;
                if(read_miso()){
                        inBuf |= 0x01 ;
                }else{
                        nop();
                }
                set_clk();
                __delay_cycles(SPI_BB_CYCLE);
        }
        __delay_cycles(SPI_BB_CYCLE);
        return inBuf;
}
```

Bus SPI

```
void transfer_data_spi(int slave, unsigned char * send_buffer,
unsigned char * receive_buffer , unsigned char length){
	unsigned int i ;
	clr_ss(slave);
	delay(N);
	for(i = 0 ; i < length ; i ++){
		receive_buffer[i] = transfer_byte_spi(send_buffer[i]);
	}
	delay(N);
	set_ss(slave);
}
```

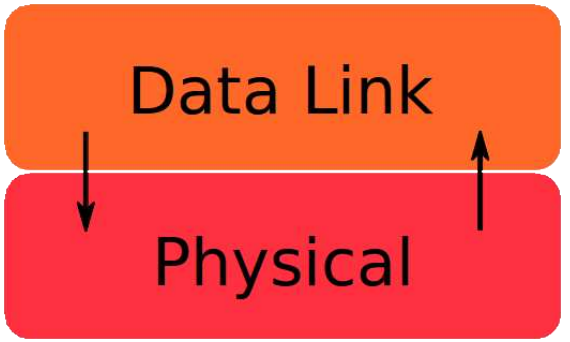
Bus SPI

- Exemple de périphériques:
  - EEPROM
  - Carte SD (mode SPI au démarrage)
  - Écran graphique
  - Convertisseur DAC, ADC
  - Capteurs inertiels (gyro, accelero)

Bus I2C

- I2C (prononcer “I square see”)
- Parfois appelé TWI

  (Two Wire Interface)
- Carte d'identité:
  - Bus série synchrone
  - Topologies supportées:
    - Point → multi-points en maître/esclaves
      - Multi-maîtres possible mais plus complexe
  - Half-duplex

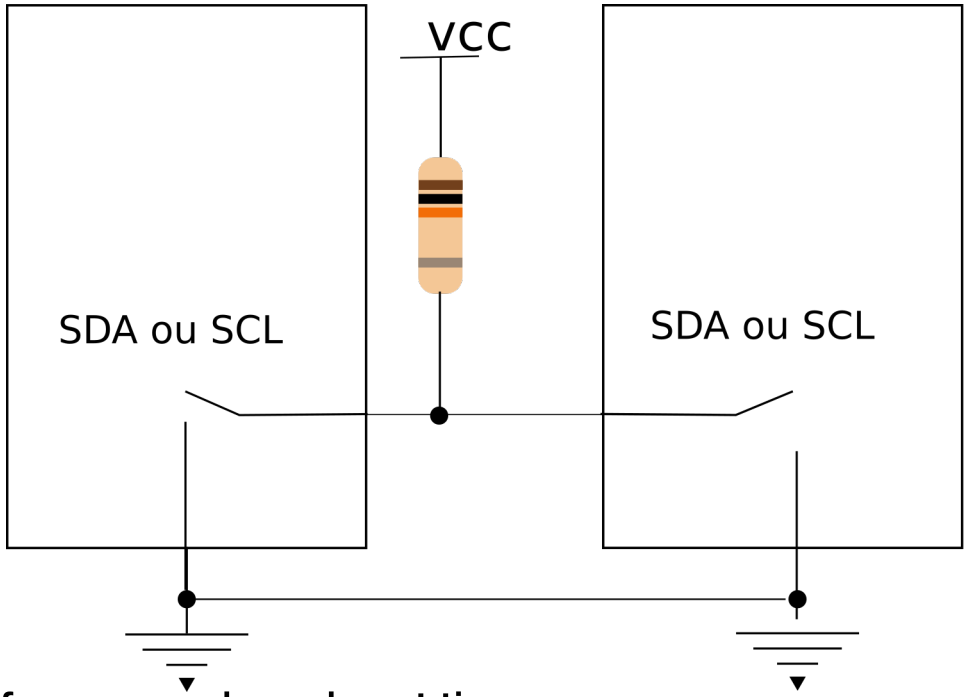


Bus I2C

- Spécifications électriques:
  - Bus utilisant des sorties à collecteur ouvert (open drain):
    - deux niveaux logiques : 0 et Hiz (haute impédance)
- Câblage:
  - 3 conducteurs : SCL, SDA, GND
- Vitesse:
  - 100KHz (normal), 400 kHz (fast-mode), 1MHz (fast-mode plus), 5MHz (ultra fast-mode)
- Longueur de lignes:
  - Bus de carte et bus de terrain
  - 100m max, dépend de l'impédance du câblage

Bus I2C

Gestion du caractère bidirectionnel des lignes de communication:



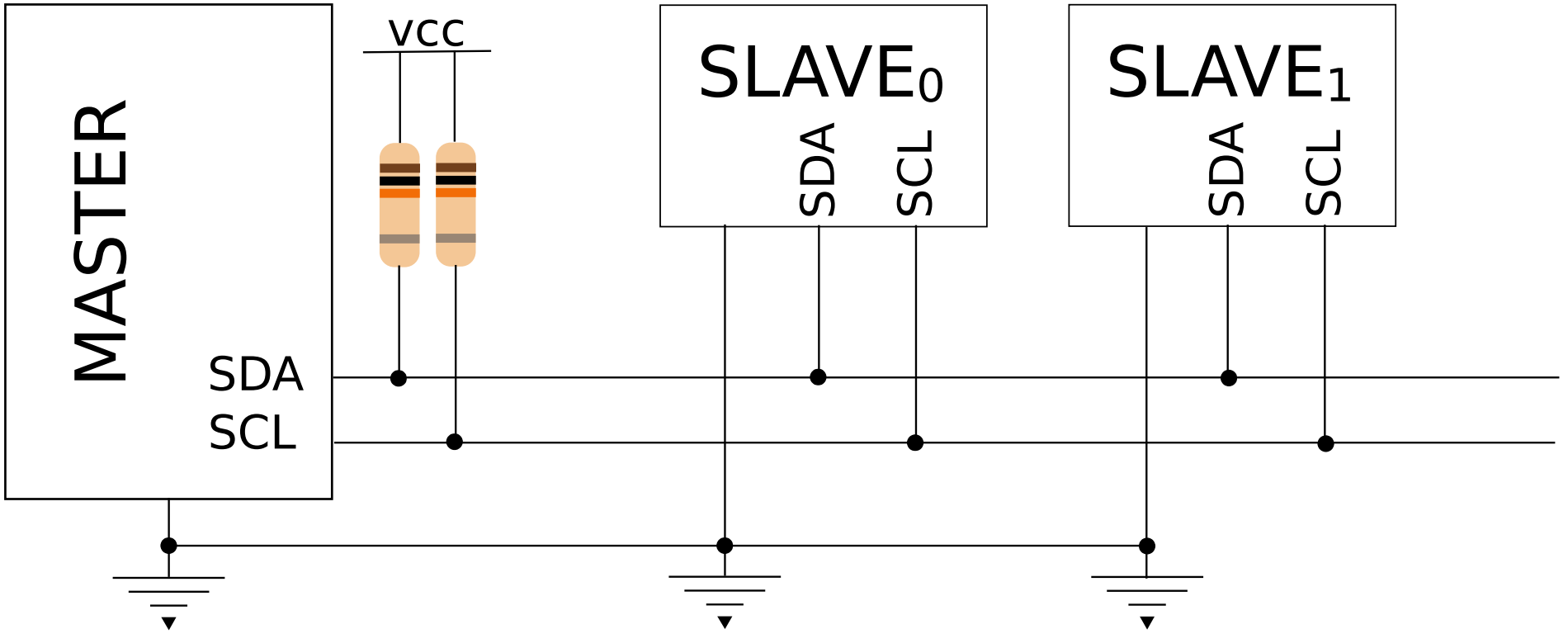
État récessif sur une broche: Hiz

Si tous les composants sur la ligne imposent l'état récessif,
la resistance de pull up amène la ligne à l'état haut

État dominant sur une broche et donc sur le bus: 0

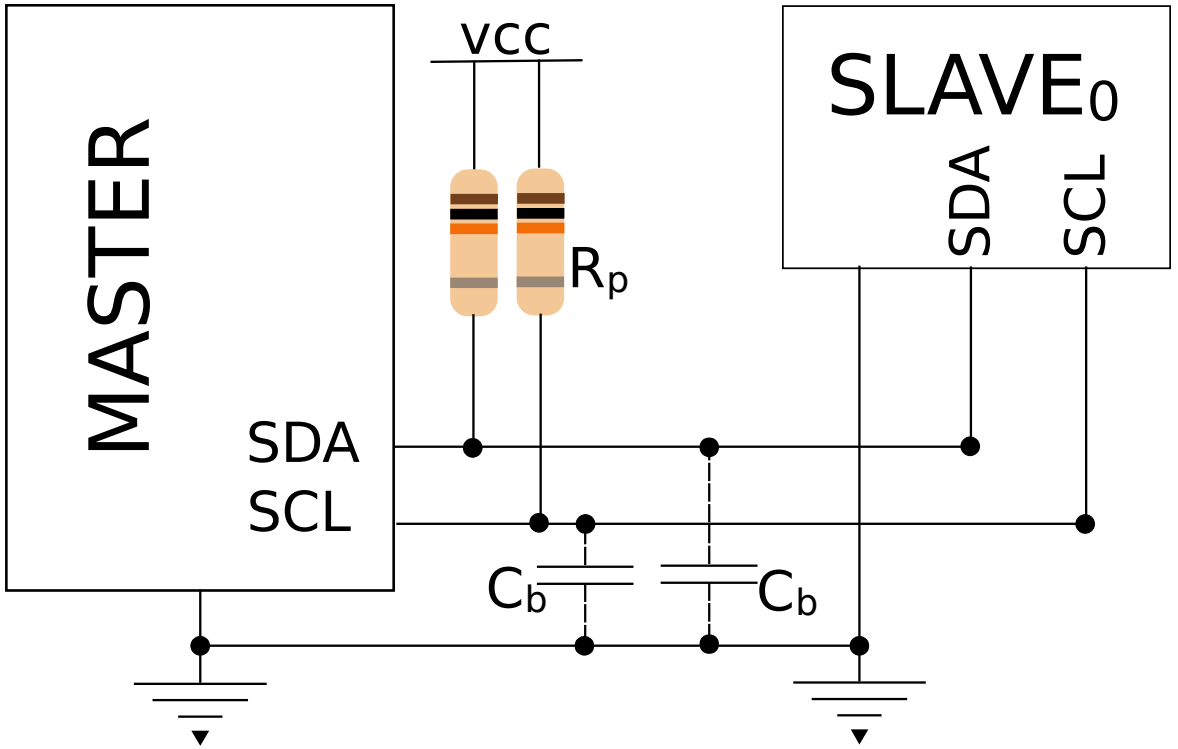
Il suffit d'un composant imposant l'état 0 sur le bus pour gagner
Le pull-up limite le courant

Bus I2C



SCL en sortie sur le maître et en entrée sur les esclaves

Bus I2C, Résistances de pull-up



Cb : Capacité du bus ~ 400pF
Rp : Résistance de pull-up

Le calcul de la résistance de pull-up est un compromis entre :

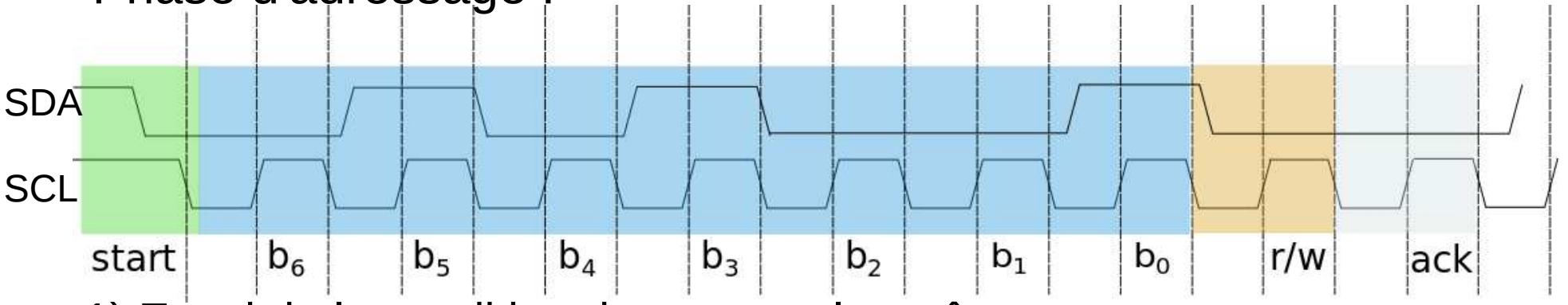
- Le courant maximum pour SCL et SDA au niveau bas
- Le temps de montée des signaux sur le bus (le couple Rp + Cp agissant comme un filtre passe bas.

Valeurs usuelles, 10kohm pour de l'i2c classique, 4.7kohm pour de l'i2c rapide.

Bus I2C

Une transaction I2C se compose d'une phase d'adressage et de 0 à n phases d'échange de données, puis d'une condition de STOP

Phase d'adressage :



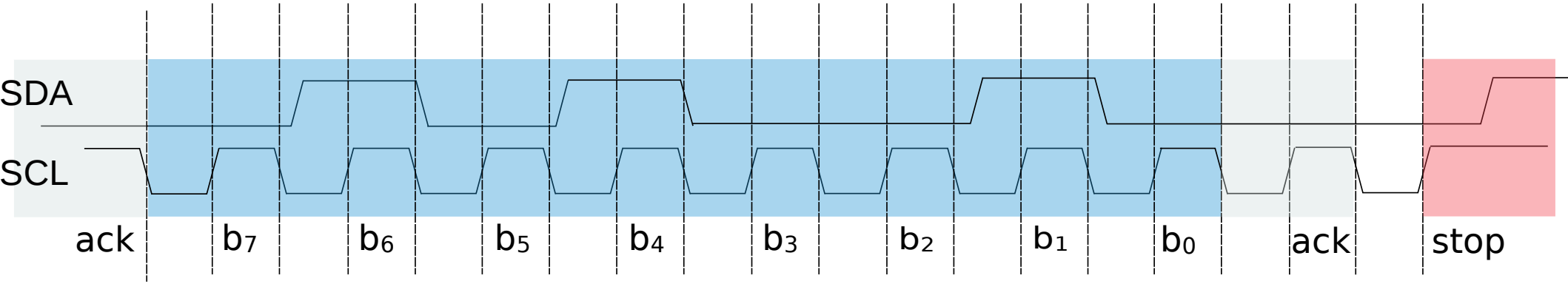
1) Envoi de la condition de start par le maître
2) Envoi des 7 bits d'adresse
3) Envoi du bit de de choix de lecture (bit à 1) ou d'écriture (bit à 0)
4) Réception de l'acquittement: le maître relâche le bus de donnée (état haute impédance) et lit le niveau du bus sur le prochain niveau haut de l'horloge. Dans le cas où un esclave correspondrait à l'adresse envoyée, celui-ci forcera le bus au niveau bas pour indiquer sa présence et/ou le fait qu'il est prêt.

I2C, Adressage matériel

- Adressage pour choisir un composant sur le bus:
  - 7 bits d'adresse (128-2 périphériques adressables logiquement en théorie)
  - 1 adresse de lecture avec bit R/W à 1
  - 1 adresse d'écriture avec bit R/W à 0
  - Attention l'adresse fournie par le constructeur peut être l'adresse :
    - 8 bits avec lecture
    - 8 bits avec écriture
    - 7 bits sans R/W
    - Il faut lire la documentation et regarder les chronogrammes pour déterminer.
  - Attention, les librairies peuvent utiliser des adresses sur 7 ou 8 bits

Bus I2C

Phase(s) d'envoi de données puis STOP :



1) Envoi des 8 bit de données (soit par le maître soit par l'esclave en fonction de R/W lors de la phase d'adressage)
2) Réception de l'acquittement

3)Si fin de la communication : envoi de la condition de stop.
Sinon la communication continue avec une nouvelle phase d'échange de données dans la même direction.

Bus I2C, Acquittement

- Rôle du bit d’acquittement (Acknowledge):
  - Acquittement de l'adresse : périphérique existe (scan du bus) et/ou est prêt.
  - Acquittement de la donnée écrite ou écrite, par le maître ou par l’esclave.
  - Mécanisme d'auto-incrémentation d'adresse logique (voir page suivante): Lors de l’acquittement pour une lecture ou une écriture vers une mémoire, l'adresse est automatiquement incrémentée pour pouvoir enchaîner avec une autre lecture ou écriture.

I2C, Adressage logique

- Adressage d'un registre dans le composant :
  - Le composant est vu comme une mémoire à plusieurs cases.
- L’adresse est envoyée comme une donnée (sur un ou plusieurs octets)
  - Pour une écriture:
    - Envoi de l'adresse du registre, puis de la/des données
  - Pour une lecture:
    - Solution 1 : Un accès en écriture puis un accès en lecture
    - Solution 2 : Renvoi de la condition de start sans condition de stop (SMBUS)

I2C, Périphérique intégré

- Exemple TI MSP430

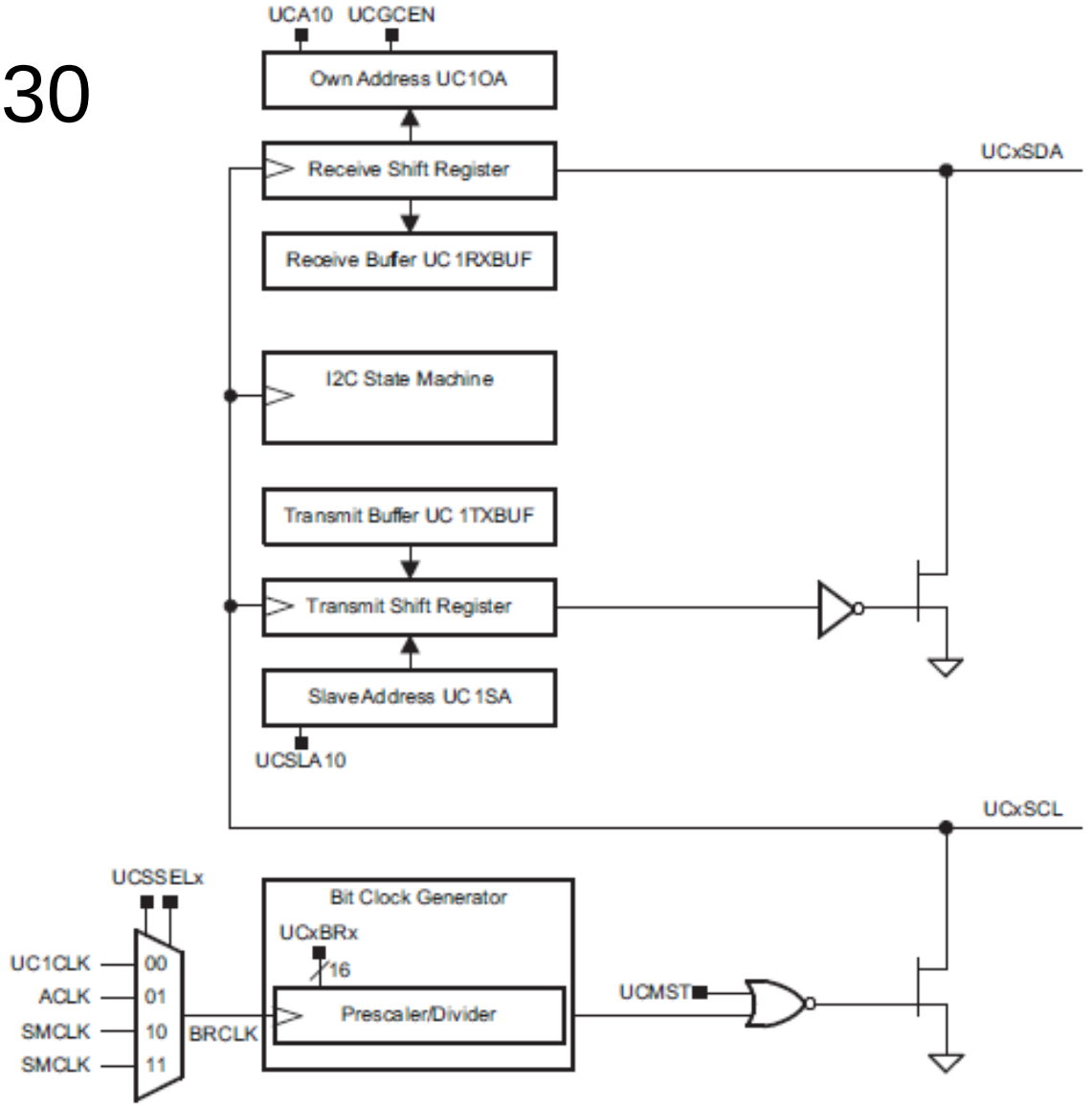


Figure 17-1. USCI Block Diagram: I²C Mode

I2C, Périphérique intégré

```
void open_i2c(){
	P1SEL |= BIT6 + BIT7 ; // configuration des IOs
	P1SEL2 |= BIT6 + BIT7 ;
	P1DIR |= BIT6 + BIT7;
	UCB0CTL1 |= UCSWRST ; //maintien en reset
	UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC ; //i2c master
	UCB0CTL1 = UCSSEL_2 + UCSWRST ; // selection de l'horloge
	UCB0BR0 = 10 ; // diviseur de l'horloge
	UCB0BR1 = 0;
	UCB0I2CSA = 0x00 ;
	UCB0I2CIE = UCNACKIE + UCALIE ; // Nack génère une interruption
	UCB0CTL1 &= ~UCSWRST ; // désactivation du reset
	IE2 |= UCB0TXIE | UCB0RXIE ; // enable des interruptions au niveau général
}
```

I2C, Périphérique intégré

```
void write_i2c(unsigned char addr, unsigned char * data, unsigned char nbData){
	UCB0I2CSA = addr ;
	i2cBusy = 0 ;
	i2cBufferPtr = data ;
	i2cBufferLength = nbData ;
	txDone = 0 ;
	UCB0CTL1 |= UCTR + UCTXSTT;
	while(txDone == 0) ;
}
unsigned char readi_2c(unsigned char addr, unsigned char * data, unsigned char nbData){
	UCB0I2CSA = addr ;
	i2cBusy = 0 ;
	i2cBufferPtr = data ;
	i2cBufferLength = nbData ;
	rxDone = 0 ;
	UCB0CTL1 &= ~UCTR;
	UCB0CTL1 |= UCTXSTT;
	if(nbData == 1){
		while(UCB0CTL1 & UCTXSTT);
		UCB0CTL1 |= UCTXSTP ;

	}
	while(rxDone == 0) ;
	return rxDone ;
}
```

I2C, Périphérique émulé

- SCL :
  - GPIO en sortie pour pour une interface maître
  - GPIO en entrée pour une interface esclave
- SDA :
  - GPIO avec buffer collecteur ouvert
    - Sur certains micro-contrôleurs, sortie en collecteur ouvert quand la GPIO est configurée en entrée
- Besoin des pull-up sur SDA et SCL
  - parfois présentes sur les périphériques

I2C, Périphérique émulé pour un maître

```
void open_i2c(){
        set_sda();
        set_scl();
}

void start(){
        clr_sda();
        delay(N/2);
        clr_scl();
}

Void stop(){
        clr_scl();
        clr_sda();
        delay(N);
        set_scl();
        delay(N/2);
        set_sda();
}
```

```
bit read_ack(){
        bit a ;
        clr_scl();
        set_sda();
        delay(N/2);
        set_scl();
        a = read_sda();
        delay(N/2);
        return a;
}

void write_byte(uchar d){
        uint I ;
        for(i = 0; i < 8 ; i ++){
                clr_scl();
                if(d & 0x80) set_sda();
                else clr_sda();
                d = d << 1 ;
                delay(N/2);
                set_scl();
                delay(N/2);
        }
}
```

```
uchar read_byte(){
        uint I ;
        uchar d = 0;
        for(i = 0; i < 8 ; i ++){
                d = d << 1 ;
                clr_scl();
                set_sda();
                delay(N/2);
                set_scl();
                d |= read_sda();
                delay(N/2);
        }
        return d ;
}
```

I2C, Périphérique émulé pour un maître

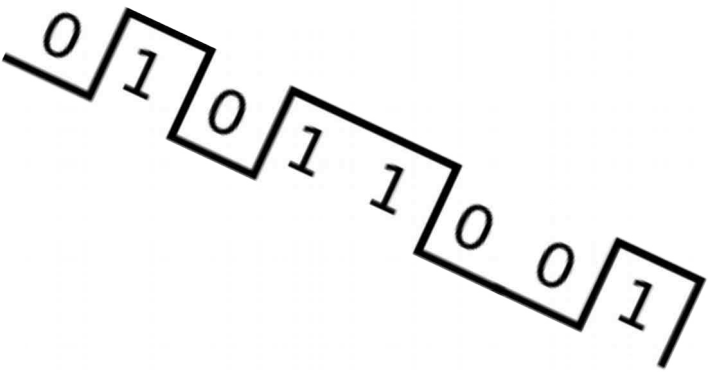
```
bit write_data(uchar addr, uchar * data, uint nbData){
	start();
	write_byte(addr << 1);
	If(read_ack()!=0){
		stop();
		return 0 ;
	}
	for(i = 0 ; i < nbData ; i ++){
		write_byte(data[i]);
		If(read_ack()!=0){
			stop();
			return 0 ;
		}
	}
	stop();
	return 1 ;
}
```

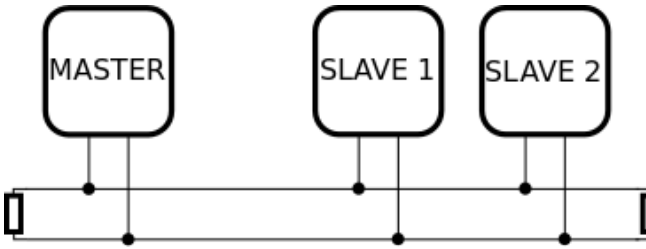
```
bit read_data(uchar addr, uchar * data, uint nbData){
	start();
	write_byte((addr << 1) | 0x01);
	If(read_ack()!=0){
		stop();
		return 0 ;
	}
	for(i = 0 ; i < nbData ; i ++){
		data[i] = read_byte();
		read_ack();
	}
	stop();
	return 1;
}
```

I2C, Exemples de périphériques

- GPIO Expander
- Horloge temps réel (RTC)
- EEPROM
- HDMI/VGA (I2C pour configurer l'écran)
- Sonde de température
- Capteurs inertiels

Bus parallèle asynchrone:
Bus Centronics
Bus VME
Bus PC104





Bus parallèle

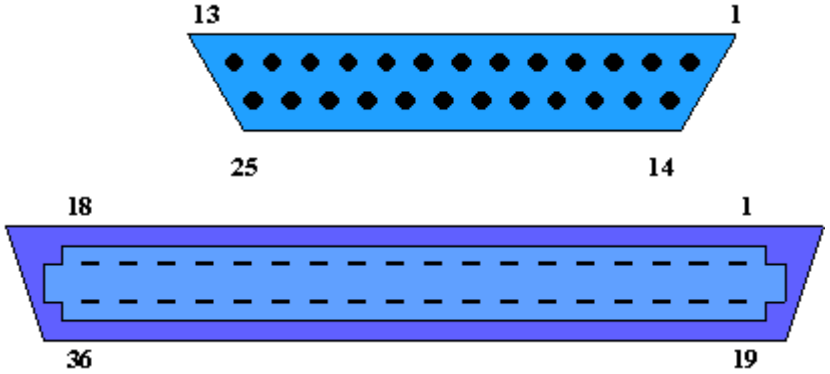
- Le bus transmet les bits d'un mot en parallèle
- Les hôtes communiquent quand ils le veulent/peuvent.
- Avantage:
  - Vitesse du bus
- Inconvénients:
  - Câblage (nombre de conducteurs et distance)
  - Sensible au bruit

Bus Parallèle Centronics

- Cartes d'identité:
  - Bus parallèle asynchrone
  - Topologie supportée:
    - Point → point en maître esclave
  - Simplex/half-duplex(IEEE 1284)

Bus Parallèle Centronics

- Spécification électrique :
  - Niveaux TTL (5v)
- Câblage:
  - 18 broches dont masse
- Vitesse:
  - ~2 MB/s
- Longueur de lignes:
  - < 3M



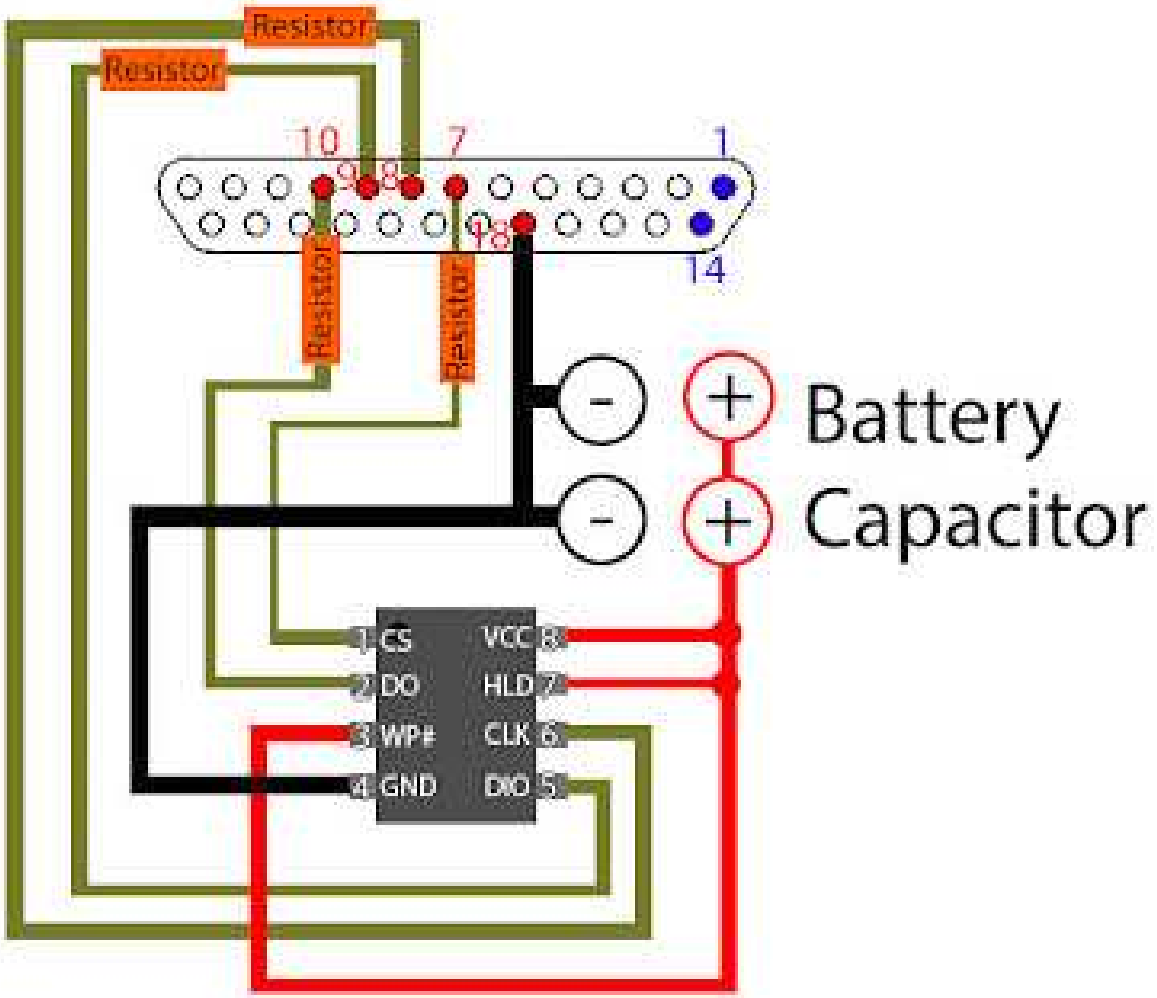
Bus Parallèle Centronics

Lignes	Numéro broche	Valeur décimale	Adresse (hexa)	Type	Etat de repos (0 Logique)
D0	2	1	378	sortie	bas
D1	3	2	378	sortie	bas
D2	4	4	378	sortie	bas
D3	5	8	378	sortie	bas
D4	6	16	378	sortie	bas
D5	7	32	378	sortie	bas
D6	8	64	378	sortie	bas
D7	9	128	378	sortie	bas
STROBE	1	1	37A	sortie	haut
AUTO LINE FEED	14	2	37A	sortie	haut
INIT/RESET	16	4	37A	sortie	bas
SELECT IN	17	8	37A	sortie	haut
ERROR	15	8	379	entrée	bas
ON LINE	13	16	379	entrée	bas
PAPER EMPTY	12	32	379	entrée	bas
ACKNOWLEDGE	10	64	379	entrée	bas
BUSY	11	128	379	entrée	haut

Bus Parallèle Centronics

- Bit du port tous adressables
- Contrôle total en logiciel
- Permet d'émuler des bus de communication “lents”
  - SPI, I2C
  - LCD
  - Protocôle de programmation de composants (EEPROM etc...)

Bus Parallèle Centronics



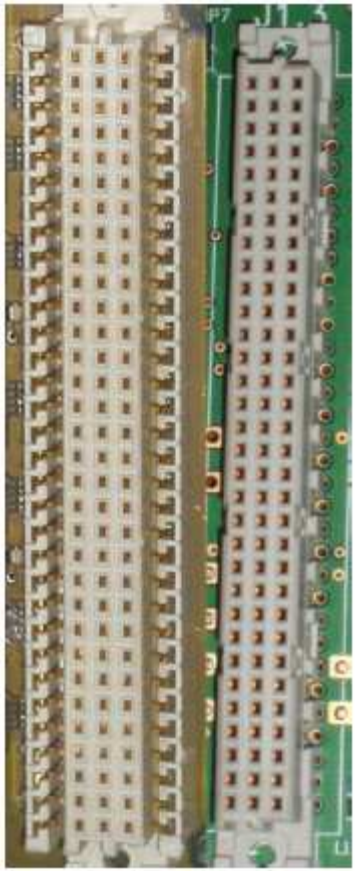
Bus Parallèle VME

- VME, Versa Module Eurocard
- Carte d'identité:
  - Bus parallèle asynchrone (pas de transmission d'horloge sur le bus)
  - Topologies supportées:
    - Point → multi-points, multi-maitres
  - Half-duplex
  - Héritage des signaux des microprocesseurs Motorola 68000

Bus Parallèle VME

- Spécifications électriques:
  - TTL 5v, haute impédance au repos
- Câblage:
  - Pas de câble, connexion carte à carte
- Vitesse:
  - 33Mhz ou 66Mhz
- Longueur de lignes:
  - Bus “Fond de panier”
- Arbitre physique sur le bus

Bus Parallèle VME, fond de panier



Bus Parallèle VME, formats de carte

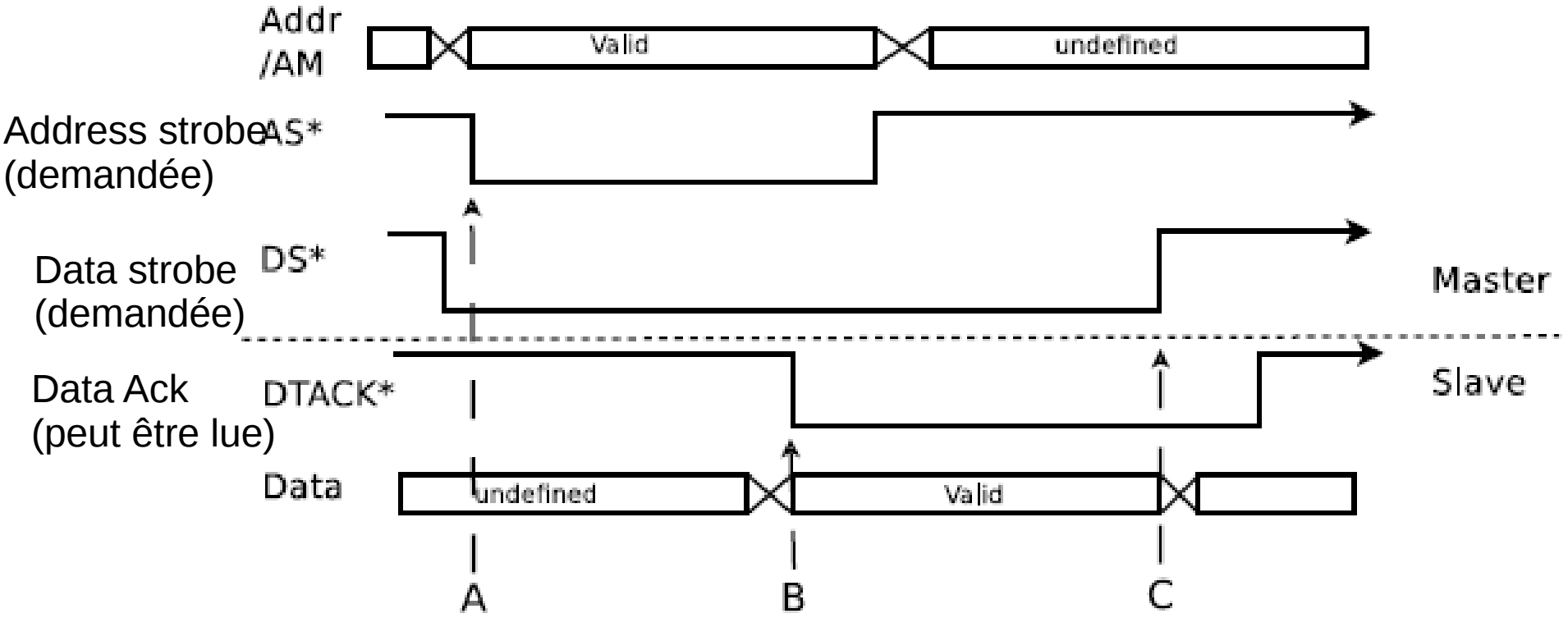
Bus Parallèle VME, Signaux

- Bus d'adresse et bus de données
  - Adresses : 16/24/32 bits
  - Données: 8/16/32bits
  - Signaux strobe pour bus de donnée et bus d'adresse. Signal Ack pour le bus de données
- Arbitrage
- Interruptions
- Défaut d'alimentation

- Détails sur: http://pen.phys.virginia.edu/daq/vme/vme-tutorial.pdf

Bus Parallèle VME, Chronogrammes

Lecture de l'esclave par le maître:



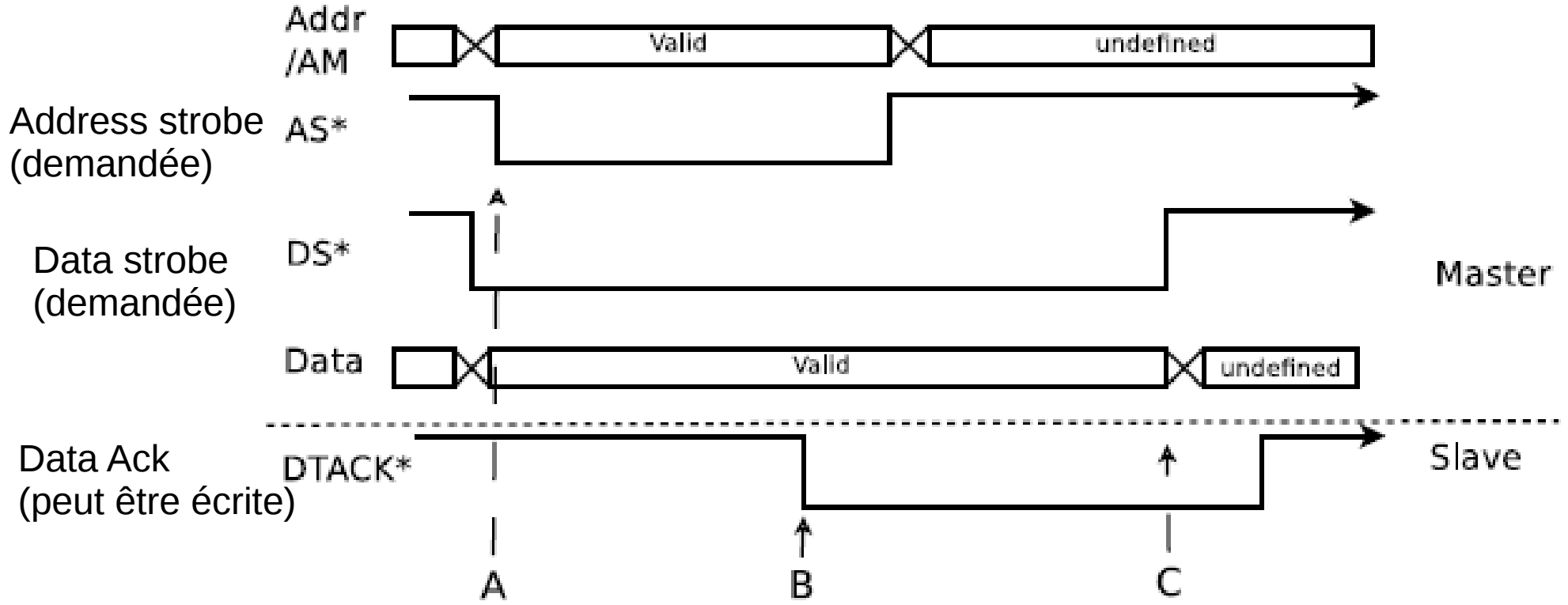
A Master requests data

B Slave provides data

C Master ends cycle

Bus Parallèle VME, Chronogrammes

Écriture vers l'esclave par le maître:



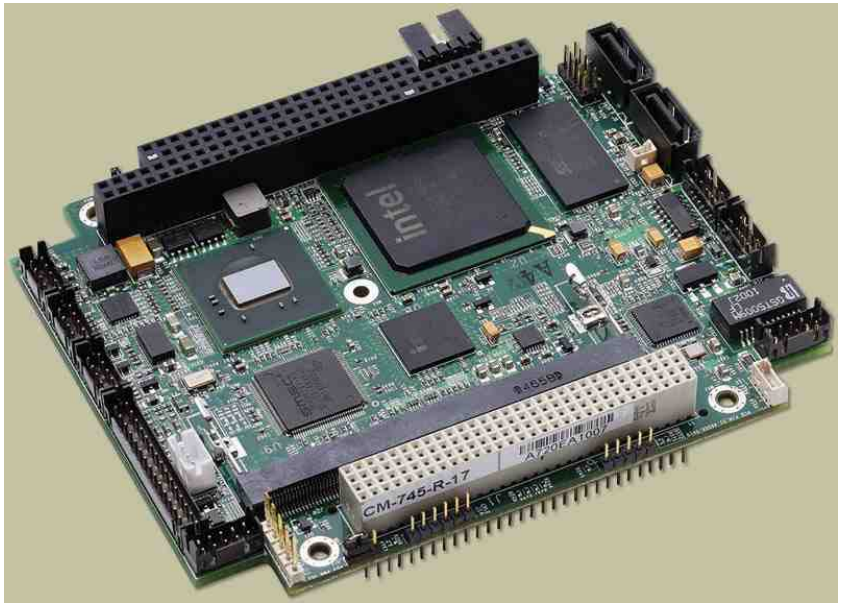
A Master provides data

B Slave accepts data

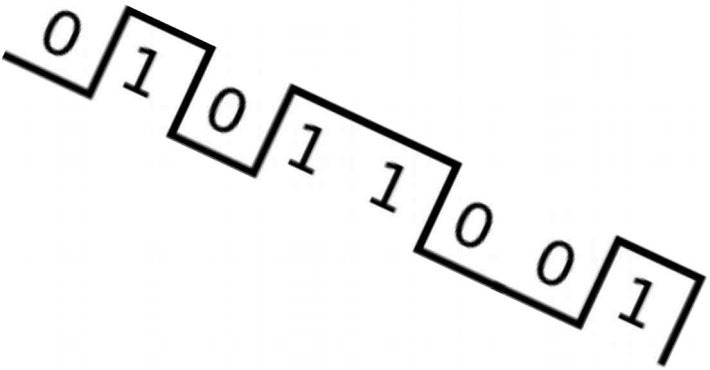
C Master ends cycle

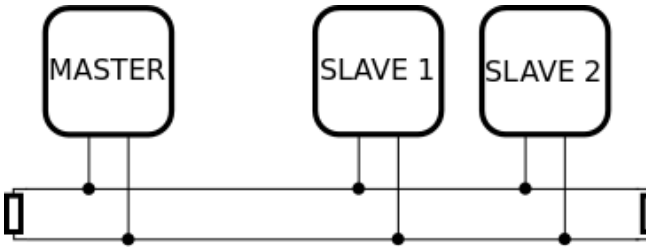
BUS PC104

- PC104: *“The stackable nature of PC/104 means that backplanes are eliminated, resulting in smaller size, lower cost, and in many cases increased ruggedness compared to alternative form factors.” (http://www.diamondsystems.com/products/pc104.php)*
- *PC104 intègre un bus parallèle ISA 8 ou 16 bits*
- *PCI-104 intègre le bus PCI*
- *PCIe/104 intègre le bus PCI Express (série)*



Protocoles sur bus série asynchrone:
NMEA (GPS)
jeu de commandes AT (GSM, Modem en général)





Système GPS

- Mire de satellites en orbite à 20 200km
- Chaque satellite transmet des messages sur les bandes de fréquences de 1.57542 GHz (L1 signal) et 1.2276 GHz (L2 signal) à une vitesse de 50 bit/s .
- Chaque message de 1500 bits (30s) contient :
  - l’information temporelle (date/heure GPS)
  - l'éphéméride (orbite du satellite)
  - l'almanach (informations de statut du satellite)

Système GPS

- Récepteur GPS
  - Triangule la position (minimum 3 satellites en vue)
  - Envoie la position à l'application sur liaison série (1 à 10Hz)
  - Format de communication le plus répandu : NMEA

Système GPS, Protocôle

- Trame NMEA
- Séquence de caractères ASCII :
  - Commence par $ suivi du nom de trame
    - GPRMC, GPGGA, GPGLL, GPGSA, GPGSV …
  - Données utiles séparées par des virgules
  - Délimiteur *
  - Somme de contrôle sur 8 bits en hexadécimal ASCII
- Trame GPRMC = Informations utiles de localisation

$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

Système GPS, Format de trames GPRMC

$GPRMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,x.x,x.x,ddmmyy,x.x,a*hh suivi de \n\r

Exemple : **$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68**

Délimiteur début de trame '$' et nom de la trame

1 = Heure universelle (UTC) du "fix"

2 = Statut du récepteur, A si le GPS fournit une information valide (fix), V sinon

3 = Latitude en degrés/minutes (llll.ll...=ddmm.mmmm) avec un nombre de

chiffres variables (de 0 à 7) après le point décimal. (dd) code l'angle en degrés (90°max) et(mm.mmmm) code les minutes en décimal (soixantièmes de degrés)..

4 = (N) Nord ou (S) Sud (équivalent au signe de la latitude ; N pour >0)

5 = Longitude en degrés/minutes (yyyyy.yy…=dddmm.mmmm). (ddd) code l'angle en degrés (180°max) et(mm.mmmm) code les minutes en décimal (soixantièmes de degrés)..

6 = (E) Est ou (W) Ouest (équivalent au signe de la longitude ; E pour >0)

7 = Vitesse au sol en nœuds

8 = Cap en degrés

9 = Date (ddmmyy)

10 = Variation magnétique du cap en degrés

11 = (E) Est ou (W) Ouest

Délimiteur fin de trame '*'

Somme de contrôle (XOR de tous les octets transmis entre $ et * EXCLUS, affichée en hexadécimal sur deux caractères ASCII)

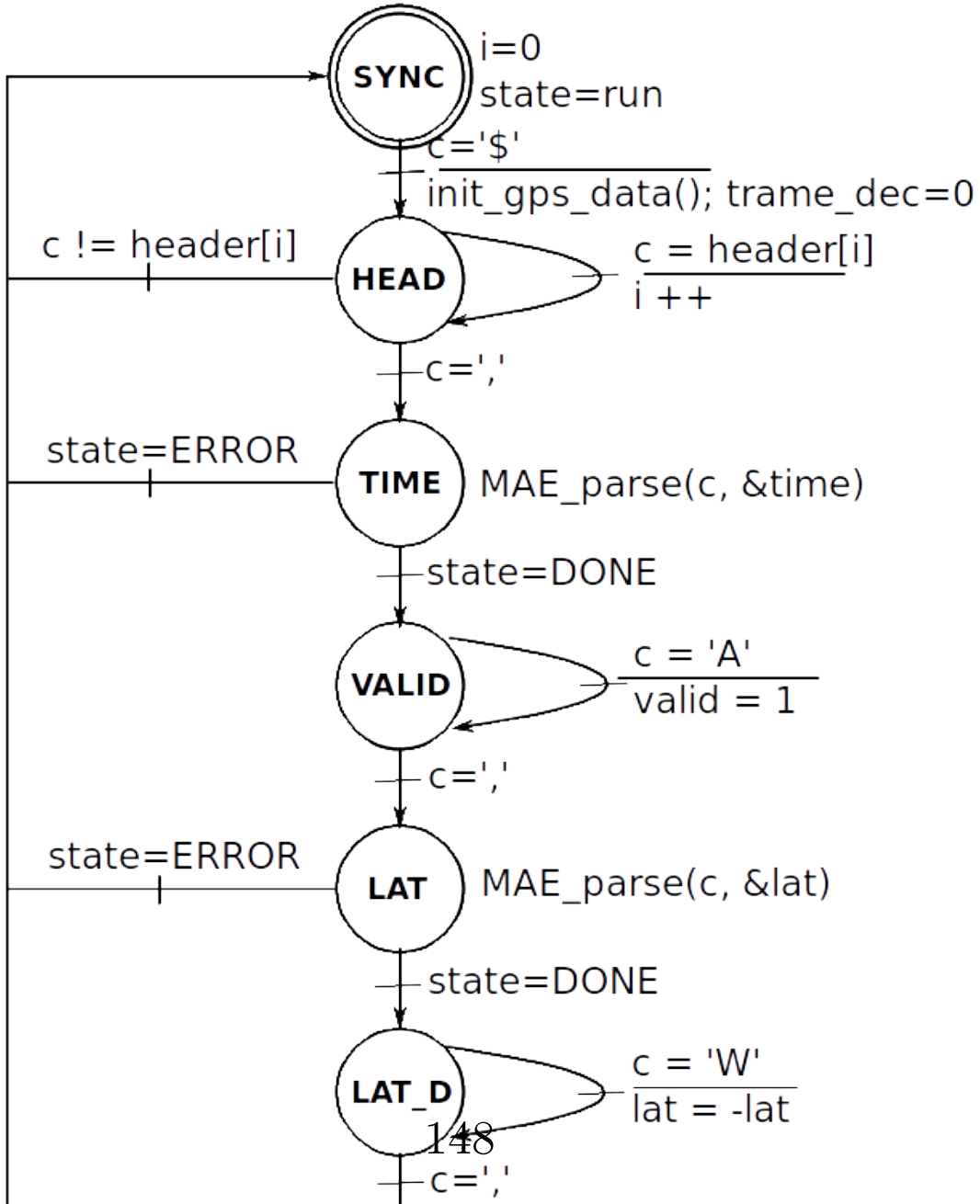
Système GPS, Décodage

- Parsing (découpage des données de la trame)
  - Buffer parsing
    - Réception des caractères dans un buffer puis traitement du buffer (consomme de la mémoire, problème de latence dans le traitement de la trame)
    - Librairies disponibles (par exemple sscanf(…))
  - Stream parsing
    - Traitement des caractères au fûr et à mesure de leur réception (peu de mémoire, pas de latence, mais fonctions plus complexes et moins réutilisables)

Système GPS, Décodage à la volée

- Stream parsing:
  - Modélisation/implémentation par machine à états
  - L'état actif symbolise le champ en cours de réception/décodage
  - Fonction non bloquante, exécutée pour chaque caractère reçu
    - Horloge de la MAE = événement « arrivée de caractère à traiter »
    - Entrée de la MAE= « caractère à traiter»
  - Le décodage doit pouvoir reprendre après une erreur (notamment retour à l'état initial puis resynchronisation sur un caractère début de trame)

Système GPS, machine à états



MODEM

- Modulateur/DEModulateur :
  - Interface entre deux domaines de communication
  - Par exemple : TTL → HF/RF

- Modem filaire pour paire cuivrée du téléphone
- Modem Courant Porteur en Ligne (injection sur le secteur)
- Modem Hyper et Radio fréquence : Modem radio 433MHz, bluetooth, GSM, GPRS, ZigBee...
- Modem optique (fibre)
- Modem audio (hydrophone)

MODEM, Commandes AT

- Données échangées en ASCII
- Préfixe de la commande par “AT” ATtention
- Certains modems nécessitent de passer en mode AT (ZigBee, Bluetooth) pour les configurer.
  - $$$ ou +++ pour passer en mode AT suivi d'une attente (qq secondes), puis le MODEM informe qu’il est passé en mode AT par OK\n

MODEM,Commandes AT pour GSM

- AT et GSM
  - Permet d'utiliser toutes les fonctions du terminal
    - Passer un appel vocal, envoi de SMS, manipulation du carnet d'adresses
  - Permet d'utiliser le mode données (GPRS ou 3G/HSDPA)
  - Utilisation de modem GSM dans le domaine M2M (Machine To Machine)

MODEM, Commandes AT pour GSM

AT+CPIN?

OK|ERROR

READY|SIM PIN|SIM PUK …

Cette commande permet de tester l'état de verrouillage du modem : ready indique que le modem est déverrouillé, SIM PIN indique que le modem est verrouillé par un code SIM, SIM PUK indique que le modem est verrouillé par un code SIM PUK. D'autres verrouillages peuvent avoir lieu (SIM PIN2 par exemple).

AT+CPIN= « 1234 »

OK|ERROR

Cette commande permet de déverrouiller le modem avec le code fourni en paramètre (ici 1234). Le modem répond **OK** si le déverrouillage a eu lieu, **ERROR** dans le cas contraire.

MODEM, Commandes AT pour GSM

- Envoie d'un SMS
  - AT+IPR=19200
  - AT+CPIN="<pin>" //unlock pin
  - AT+CREG? // get network registration
  - AT+CPMS= // select memory
  - AT+CMGF=? // get mode
  - <should be 0 PDU mode>
  - AT+CMGS=<length> // store PDU message into memory
  - ><PDU packet> <ctrl+z>
  - <return mess num>
  - AT+CMSS=<mess num> // send message
  - AT+CMGD=<messs num> //delete message

MODEM, Commandes AT pour GSM

- SMS PDU

“00”	Longueur du champ SMSC. 00 indique que le champ est complété automatiquement par le modem (préférrable)
“11”	Premier octet du message à envoyer (11 : à envoyer)
“00”	Référence du message. 00 indique que le modem doit compléter ce champ
“0B”	Longeur de l'adresse/numéro (11 chiffres)
“91”	Type de l'adresse (91 : international, 92 : locale)
“33 65 22 85 12 F0”	Numéro de téléphone, en demi-octets inversé (ici : 33 5 62 25 82 10) et terminé par F si la taille est impaire.
“00”	Identifiant de protocole (00 : SMS)
“04”	Encodage des données (00 = 7bit, 04 = 8bit, 08=16bit)
“AA”	Période de validité (AA indique 4 jours)
“07”	Taille du message en octets
“62 6F 6E 6A 6F 75 72”	Encodage en hexadécimal des octets du message (ici : «bonjour»)

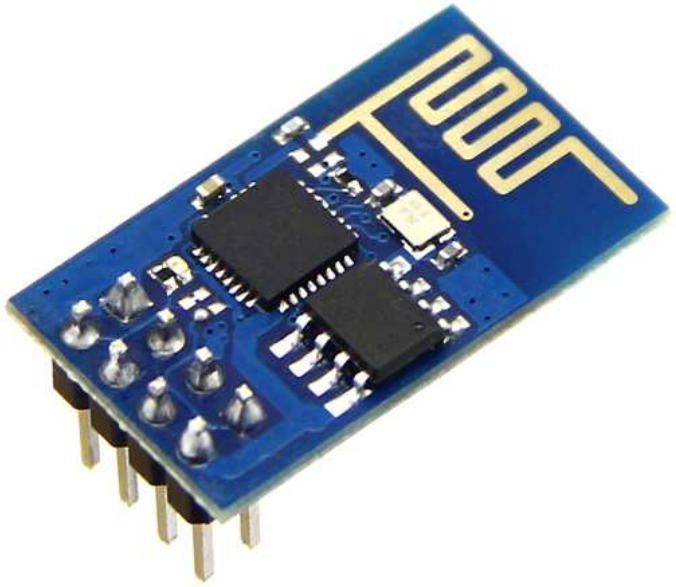
MODEM, Commandes AT pour GSM

- Codage 7-bits seulementdes caractères
- Utilisation d'octets 8 bits pour la communication
- Exemple de la chaîne "12345678" envoyée sur 7 octets :

Chars	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'
Code(hex)	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38
Septets	0110001	011001 0	01100 11	0110 100	011 0101	01 10110	0 110111	0111000
Octets	0 0110001	11 011001	100 01100	0101 0110	10110 011	110111 01	0111000 0	
Bytes	0x31	0xD9	0x8C	0x56	0xB3	0xDD	0x70	

MODEM,Modules WIFI

- ESP8266 : ~3$
  - Wifi 802.11b/gn
  - AP/SLA

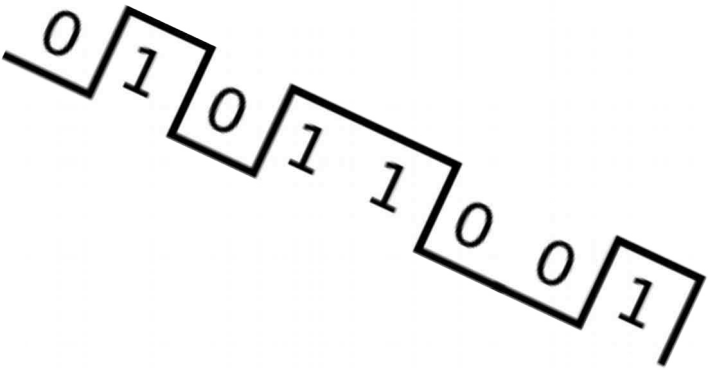


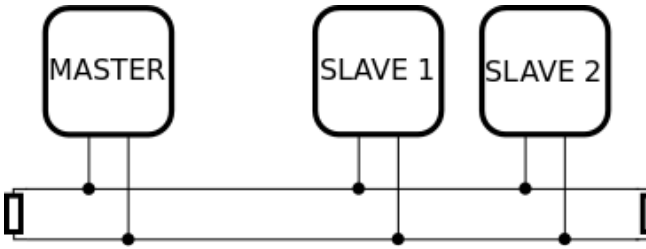
MODEM,Modules WIFI

Se connecter à une IP avec le ESP8266 :

```
issue_command("AT+RST");
uart_send_data((unsigned char *)"AT+CWJAP=", 9);
uart_send_data((unsigned char *)AP_SSID, LEN_SSID);
uart_send_data((unsigned char *)",", 1);
uart_send_data((unsigned char *)WPA_KEY, LEN_WPA_KEY);
issue_command("AT+CIPMUX=1");
issue_command("AT+CIFSR");
issue_command("AT+CIPSTART=0,\"TCP\",\"data.sparkfun.com\",80");
```

Communication optique: Standard de télécommande infrarouge Li-Fi





Communication optique

- Le média de communication est le spectre lumineux visible/infrarouge
  - Transmission « dans l'air » (ou dans le vide)
  - Transmission dans guide d'onde (fibre optique)

Communication Infrarouge (IR)

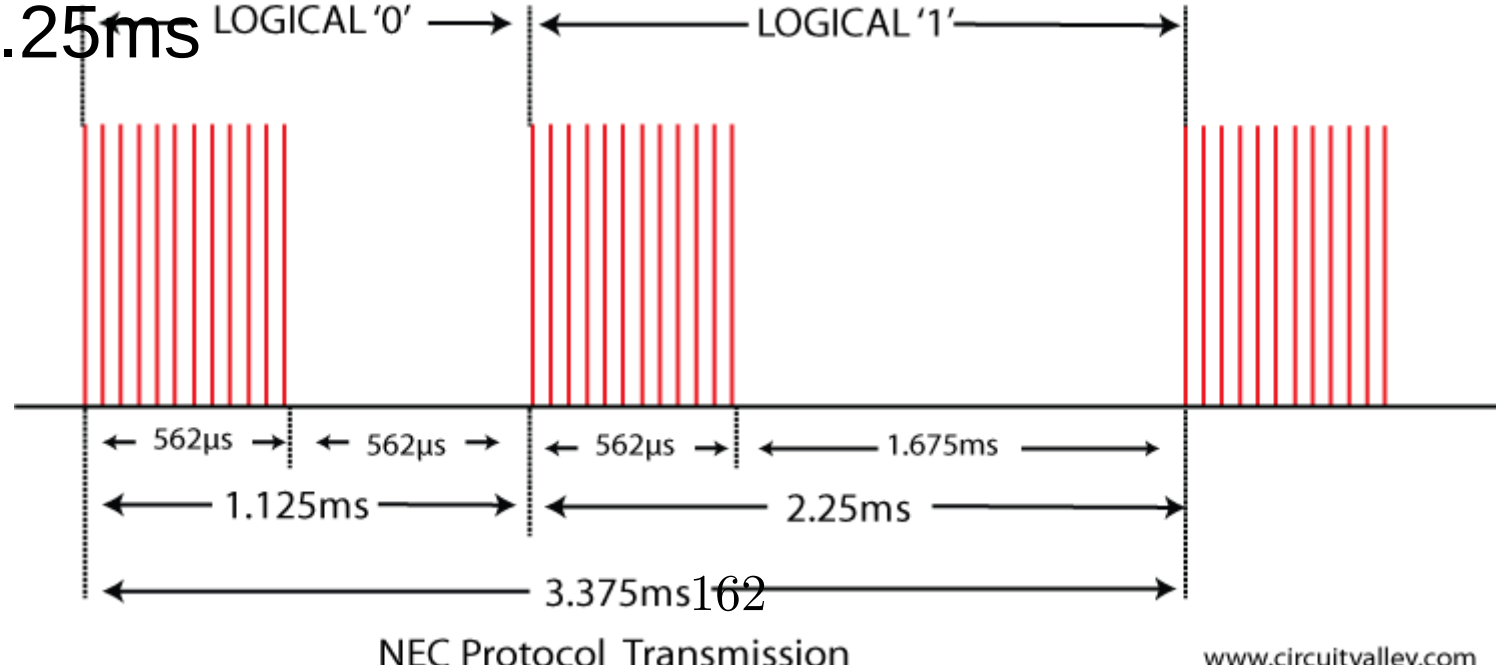
- Télécommande infrarouge (870nm, 930-950nm)
- Protocoles NEC, Sony SIRC, Philips RC5, Philips RC6 …
- Modulation d'une porteuse 36kHz (RC5/6) ou 38kHz (NEC)

Communication IR, Protocole NEC

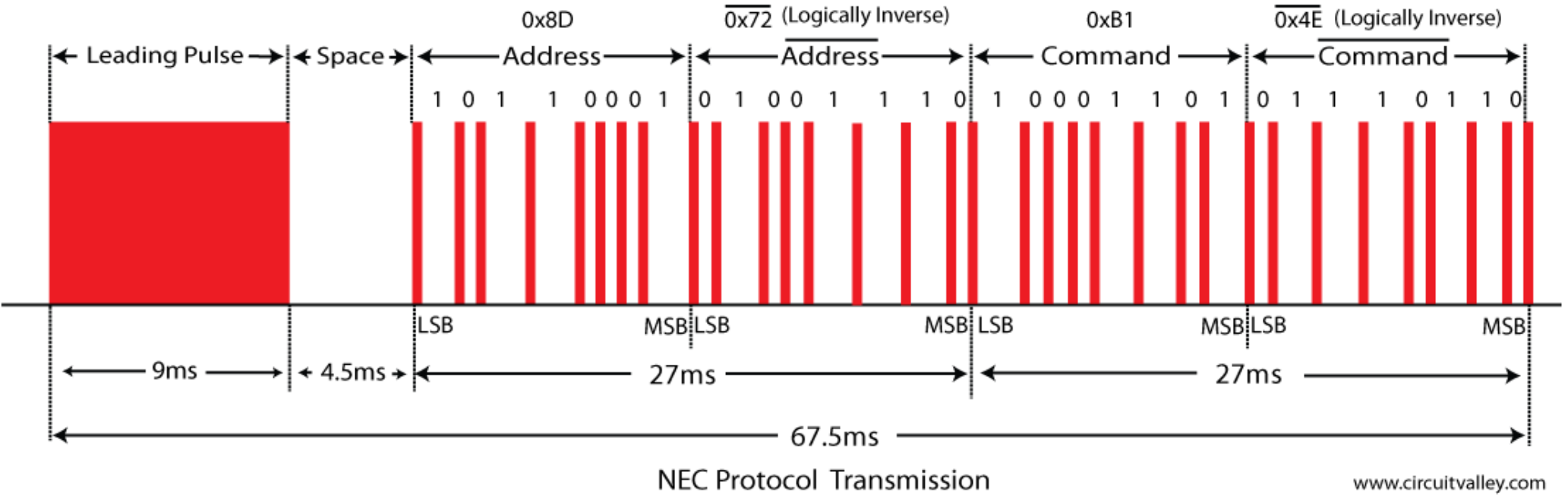
- A 9ms leading pulse burst (16 times the pulse burst length used for a logical data bit)
- A 4.5ms space
- The 8-bit address for the receiving device
- The 8-bit logical inverse of the address (8 more adressing bits in extended mode)
- The 8-bit command
- The 8-bit logical inverse of the command
- Final 562.5µs pulse burst to show end of message transmission.

Communication IR NEC

- Codage des '1' & '0'
  - Logical '0' – a 562.5µs pulse burst followed by a 562.5µs space, with a total transmit time of 1.125ms
  - Logical '1' – a 562.5µs pulse burst followed by a 1.6875ms space, with a total transmit time of 2.25ms

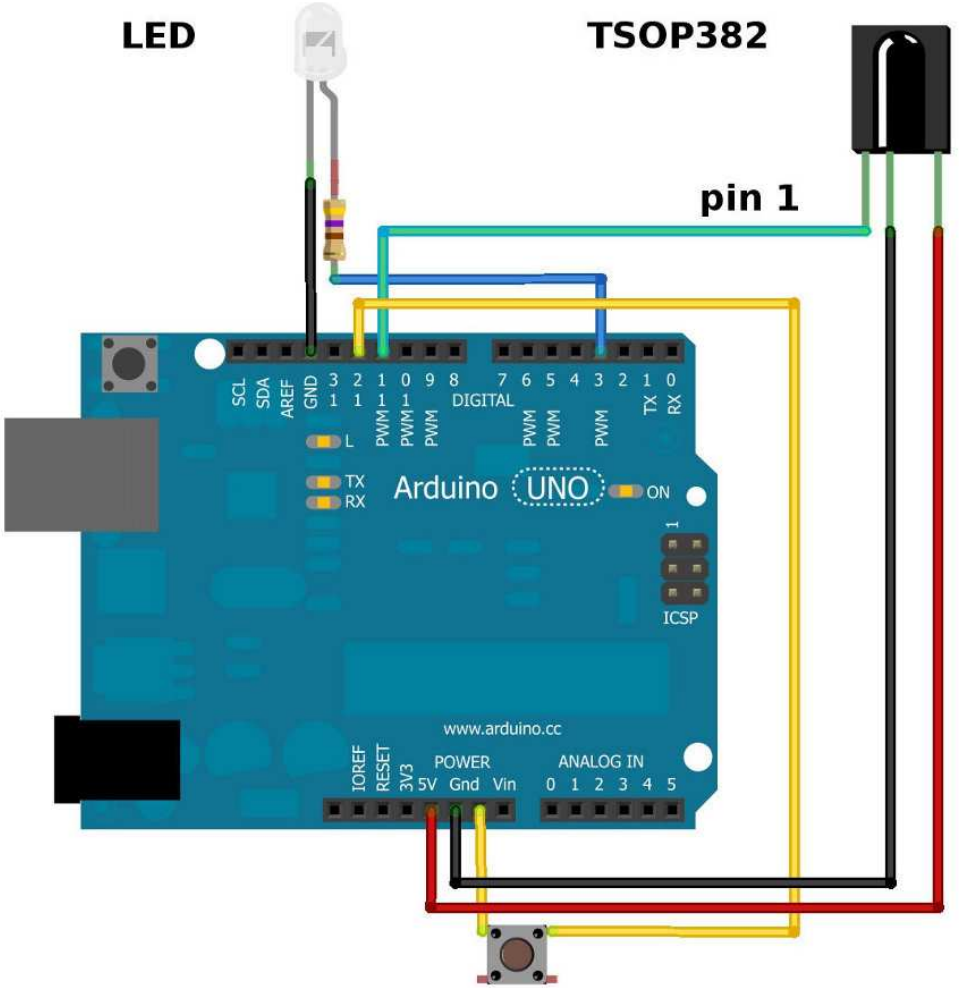


Communication IR NEC, Exemple de trame



Communication IR NEC Cablage

- Émetteur récepteur sur Arduino à réaliser en TP:
- Émission gérée par GPIO + Timer pour générer la porteuse à 38Khz
  - Le microcontrôleur doit générer le signal démodulé (enveloppe) et le signal modulé est généré par l'activation/inhibition du périphérique
- Réception gérée par récepteur/démodulateur avec filtre Passe Bande et contrôle automatique du gain (AGC) connectée à une GPIO
  - Le microcontrôleur doit décoder le signal démodulé



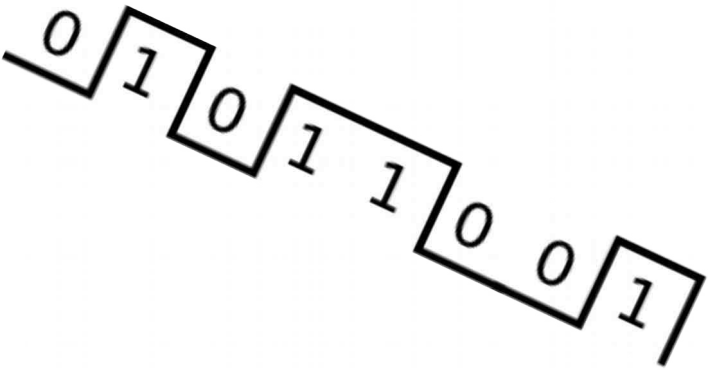
Communication IR

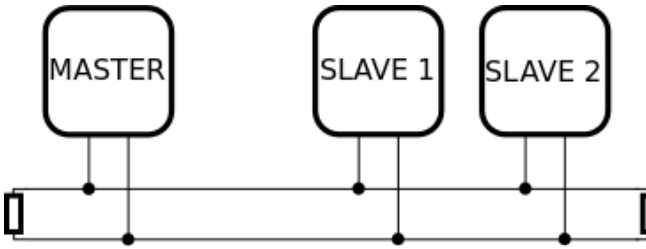
- Les codes des commandes dépendent du constructeur et du modèle :
  - NEC
  - SONY
  - RC5
  - RC6
  - ….
- Librairies plus ou moins génériques disponibles

Communication optique, Li-Fi (Light Fidelity)

- Communication par éclairage ambiant (LED) utilisant le spectre visible : Visual Light Communication
- Modulation de l'éclairage ambiant à un fréquence non perceptible à l’œil nu
  - Pas de problèmes d’attribution de fréquence au niveau législatif
  - Large gamme de fréquence disponible
  - Pas d’interférence avec radio
  - Protection vis à vis de l’écoute par des tiers (la lumière est facilement arrêtée par un élément occultant)
- Encore en développement

Communication par Modulation par Largeur d'Impulsion:
Protocole OneWire
Protocole NeoPixel





Communication MLI

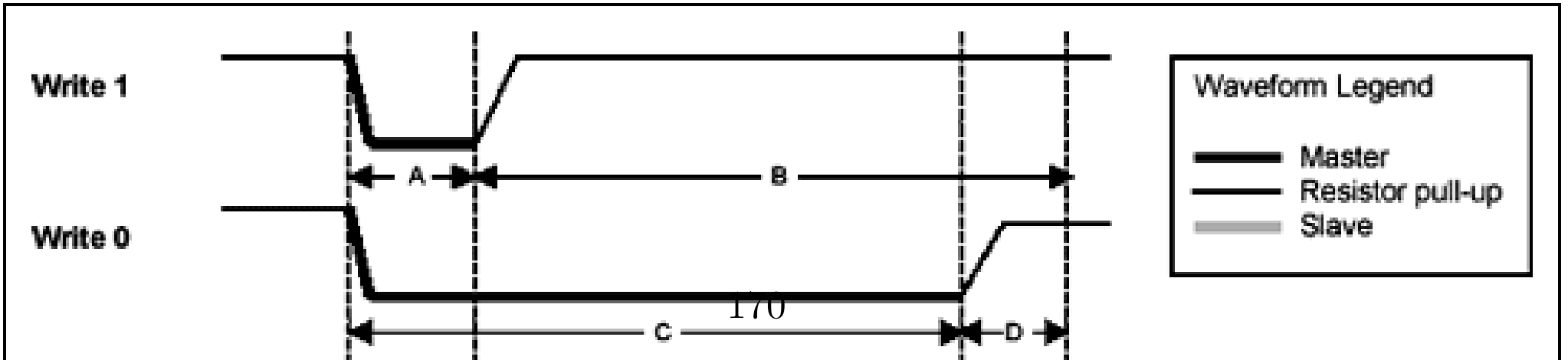
- Modulation par Largeur d'Impulsion : L'information est codée par une durée de maintien à un état ou par le rapport cyclique du signal (plutôt que par niveau)
- Avantages :
  - Pas de synchronisation à récupérer (chaque bit transmis génère un front montant et un front descendant)
  - Facilité de mise en œuvre
- Inconvénient :
  - Débit réduit par rapport à d'autres modulations

Communication MLI, Protocole OneWire

- Développé par Dallas Semiconductor
- Vitesse de communication 16.3 kbit/s
- Bus bidirectionnel et Half Duplex
- Tension de 3v ou 5v
- Possibilité de 'parasite power ' : alimentation du périphérique par les données

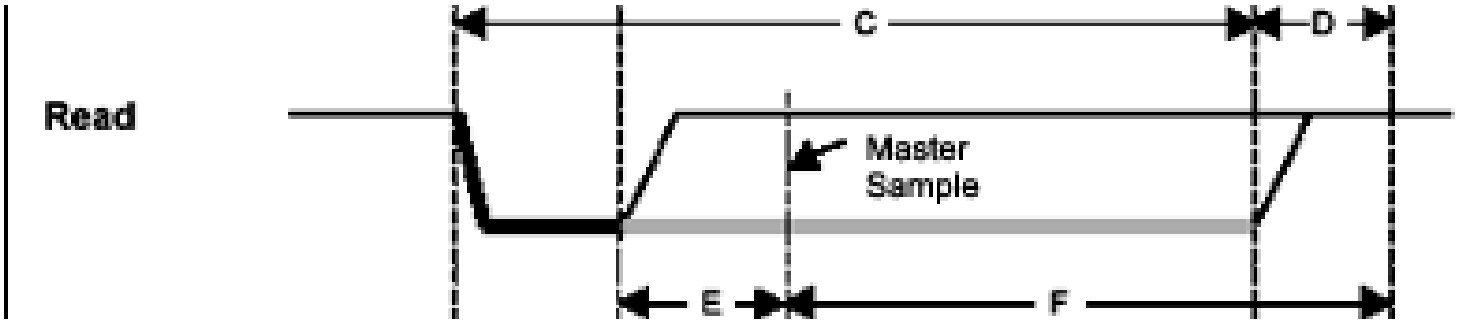
Communication MLI, Protocole OneWire

- Émission (période d'un bit de l’ordre de 60µs):
  - Émission d'un '1'
    - Mise à l'état dominant de la ligne (0v) pendant 15µs
    - Mise à l’état récessif (>1v) pendant 45µs
  - Émission d'un '0'
    - Mise à l’état dominant pendant 60µs
    - Mise à l'état récessif pendant 1µs à 15µs



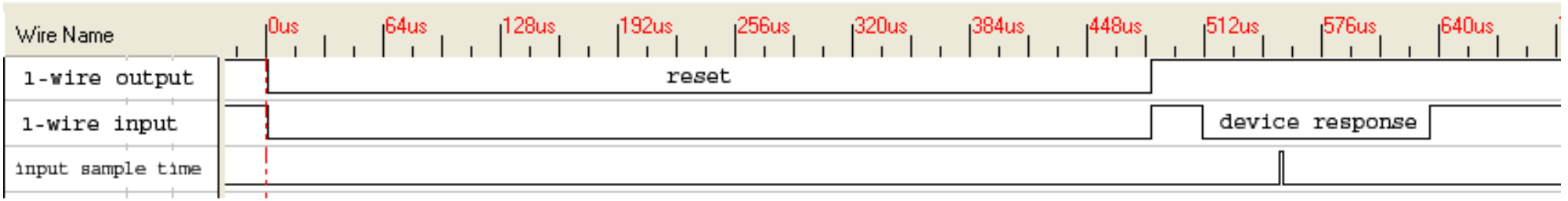
Communication MLI, Protocole OneWire

- Réception
  - Émission d'un '1' et lecture de l’état de la ligne au bout de 30µs après le front descendant

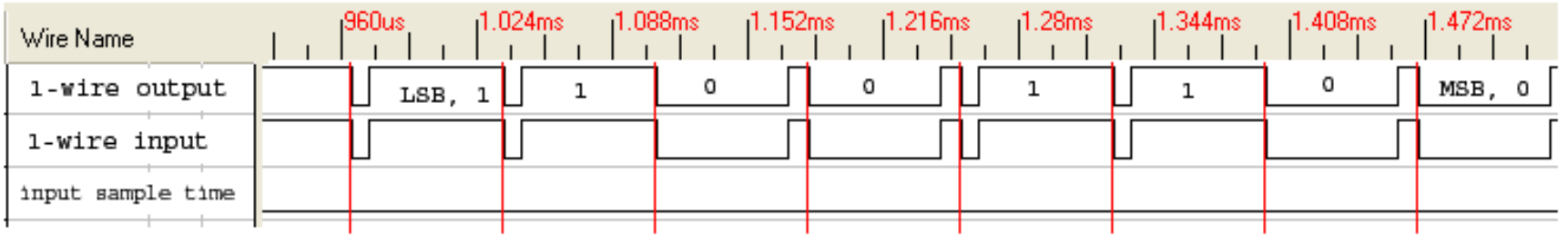


Communication MLI, Protocole OneWire

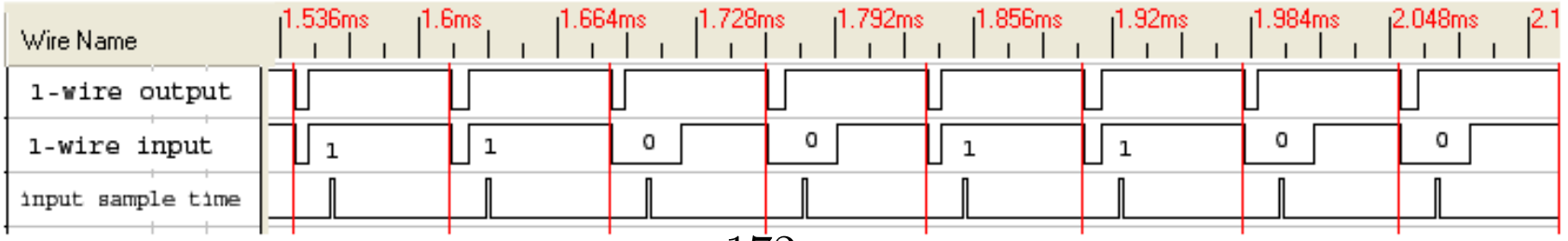
1 Wire reset, write and read example with DS2432



reset procedure



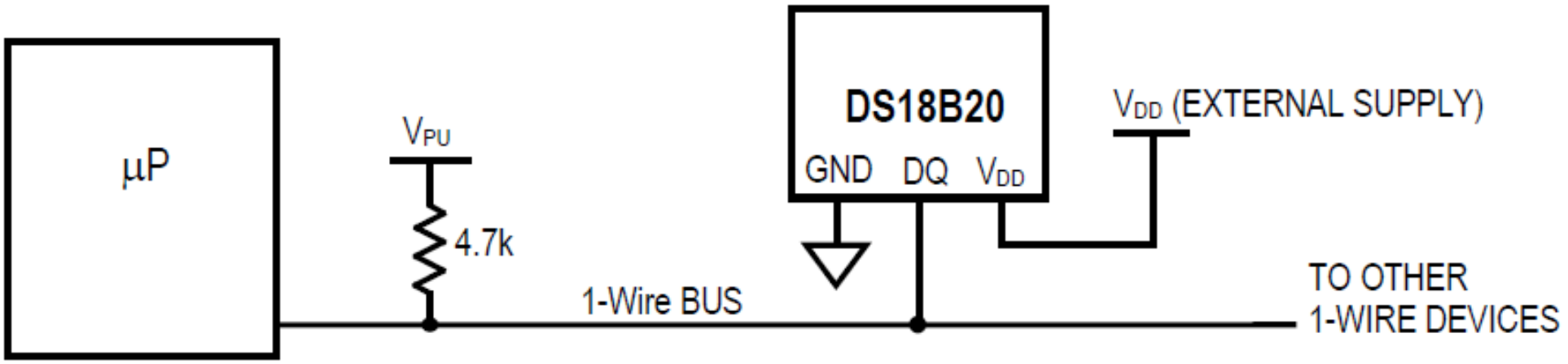
send byte x"33" (b"00110011")



read result (first byte: family code x"33")

Communication MLI, Protocole OneWire

- Exemple à traiter en TP : la sonde de température DS18B20



Communication MLI, Protocole OneWire

- Lire la sonde DS18B20
  - Générer les signaux de communication en pilotant une GPIO du microcontrôleur en mode collecteur ouvert
  - Comprendre le mécanisme d’adressage d’un composant sur le bus
  - Comprendre le mécanisme d’énumération des composants sur le bus

Communication MLI, Protocole NeoPixel

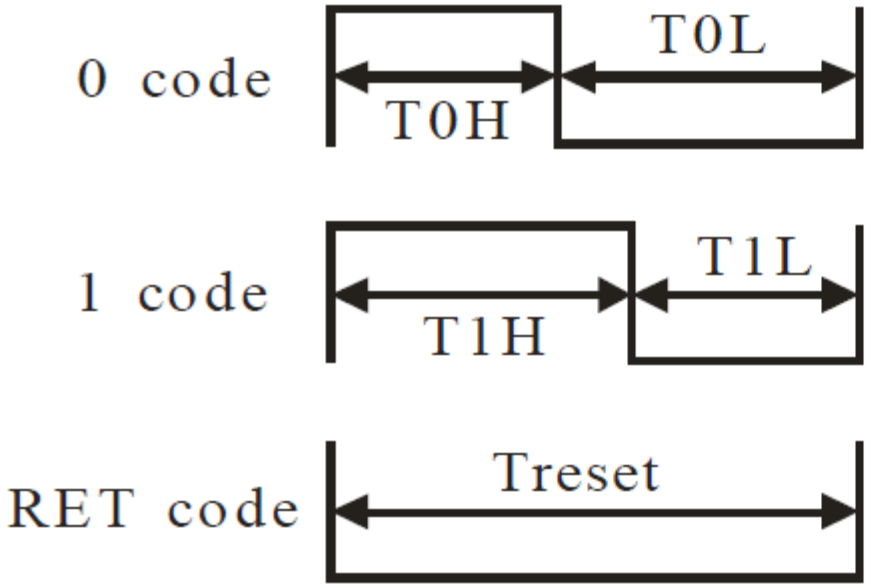
- Protocole de pilotage de LED RGB à contrôleur intégré
- Communication simplex
- Période d'un bit : ~1.30µs
- Débit max : ~800kbps
- Permet le « daisy chaining »

Communication MLI, Protocole NeoPixel

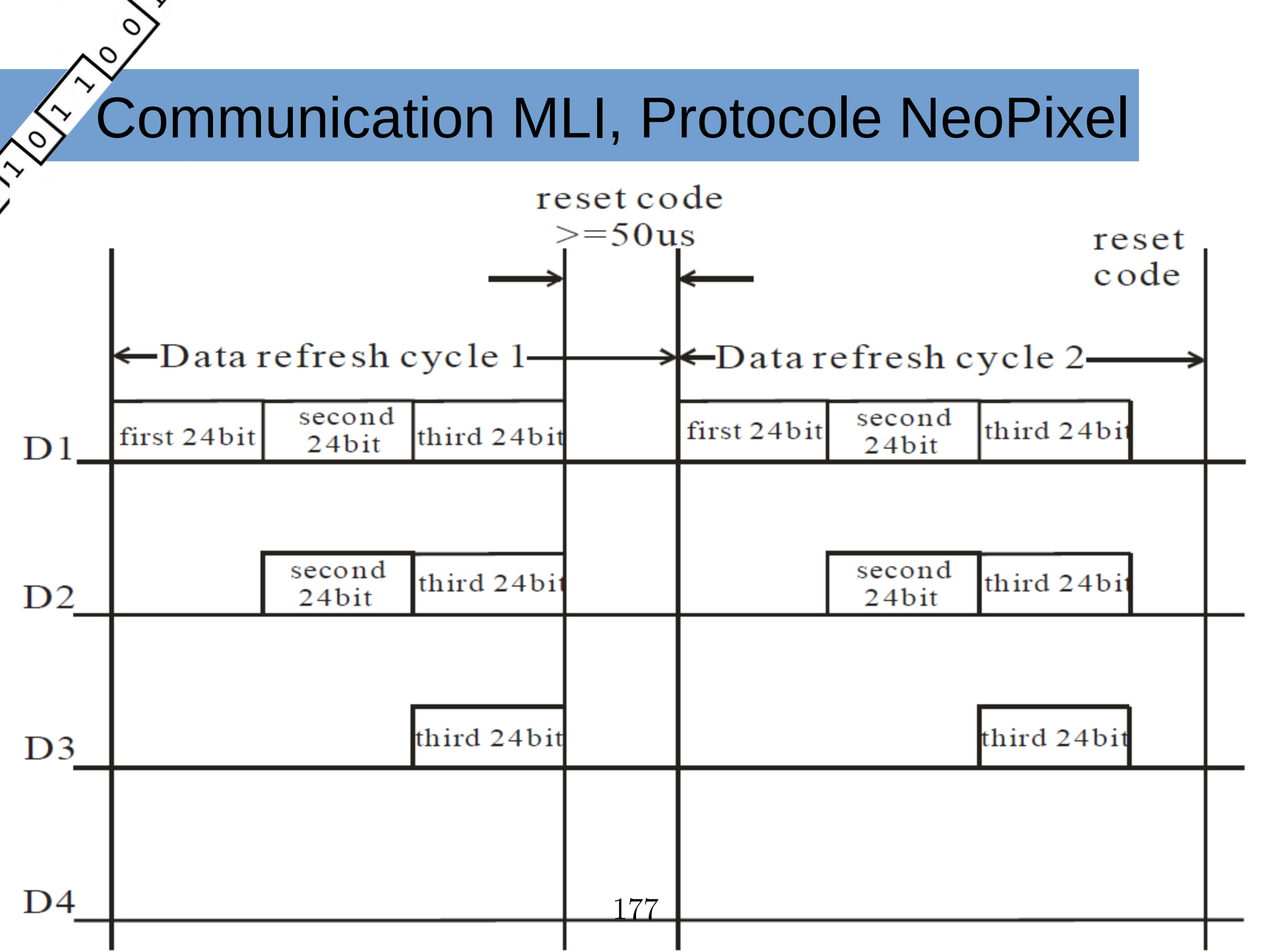
Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.7us	±150ns
T0L	0 code , low voltage time	0.8us	±150ns
T1L	1 code ,low voltage time	0.6us	±150ns
RES	low voltage time	Above 50μs	

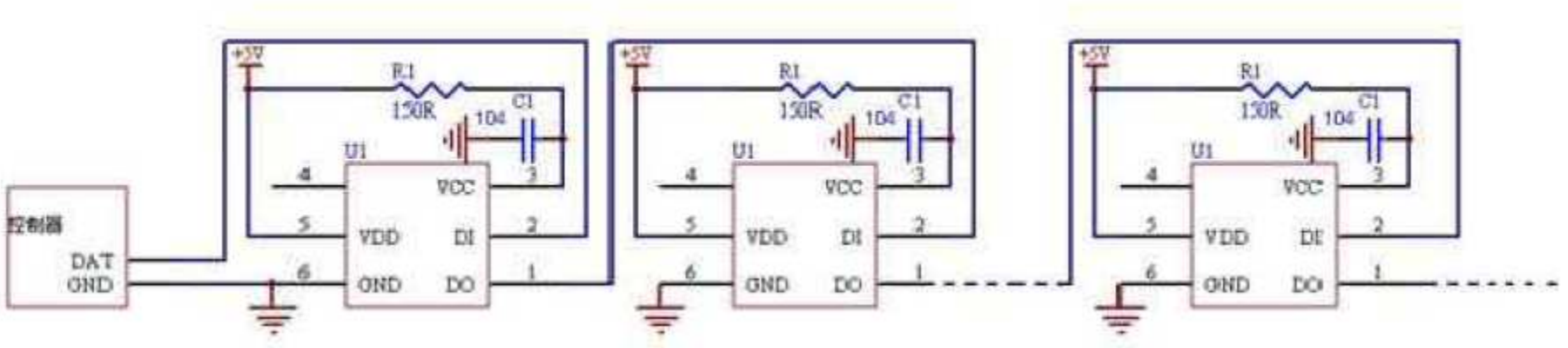
Sequence chart:



Communication MLI, Protocole NeoPixel



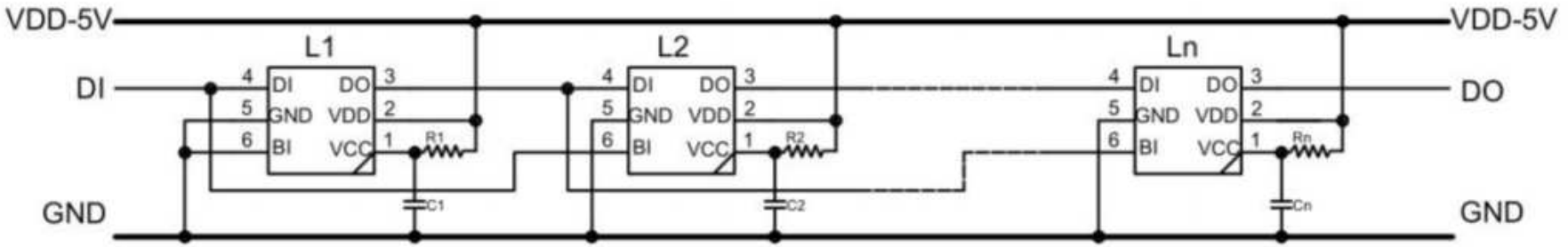
Communication MLI, Protocole NeoPixel



- Câblage Daisy Chain simple (WS2812) sensible à la défaillance d’une seule LED

Communication MLI, Protocole NeoPixel

1. Recommended application circuit



- Câblage Daisy Chain redondant (WS2813) sensible à la défaillance de deux LED successives