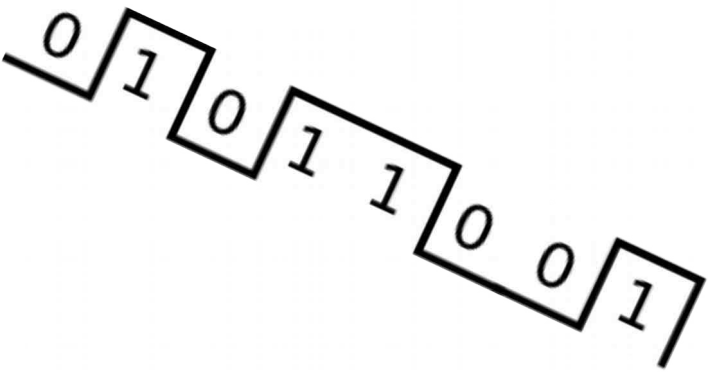
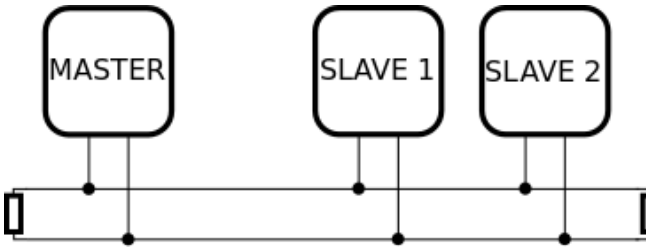
Module tronc commun : Réseau 2 et Bus de communication
Module spécialité AU : Ethernet Industriel

Seconde année Génie Électrique et Informatique Industrielle



Bertrand Vandeportaële
Jonathan Piat

Version 2018



Présentation

- Bertrand Vandeportaële et Jonathan Piat, Enseignants Chercheurs
- Robotique au LAAS
- WIKI http://homepages.laas.fr/bvandepo/wiki/
  - Polycopiés de cours
  - Sujets de Travaux Pratiques
  - Ressources Documentaires

Au menu...

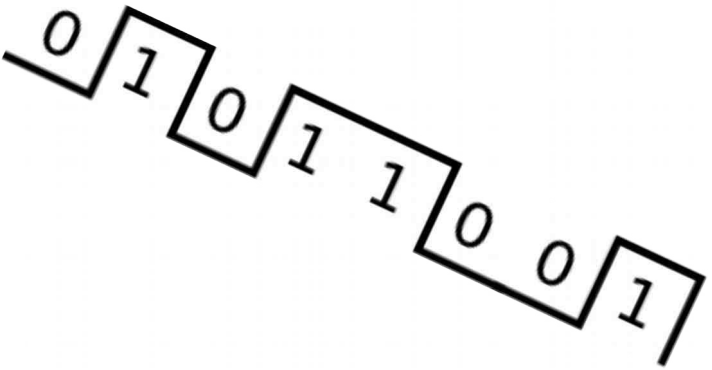
- Module tronc commun
  - Réseau 2: Mise en oeuvre d'une communication Ethernet entre un microcontrôleur PIC32 et un PC
    - Programmation Orientée Objet
    - Développement d'interface graphique avec QTCreator
    - Environnement MPLABX
  - Bus de communication:
    - Généralités sur les Bus
    -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

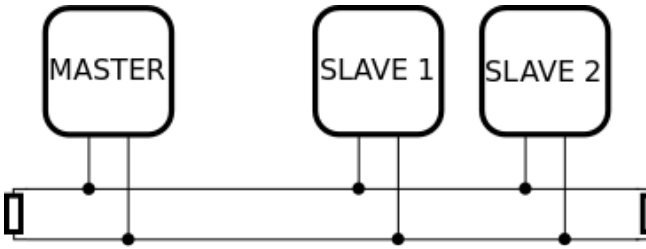
Au menu...

- 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
  - Evaluation :
    - 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



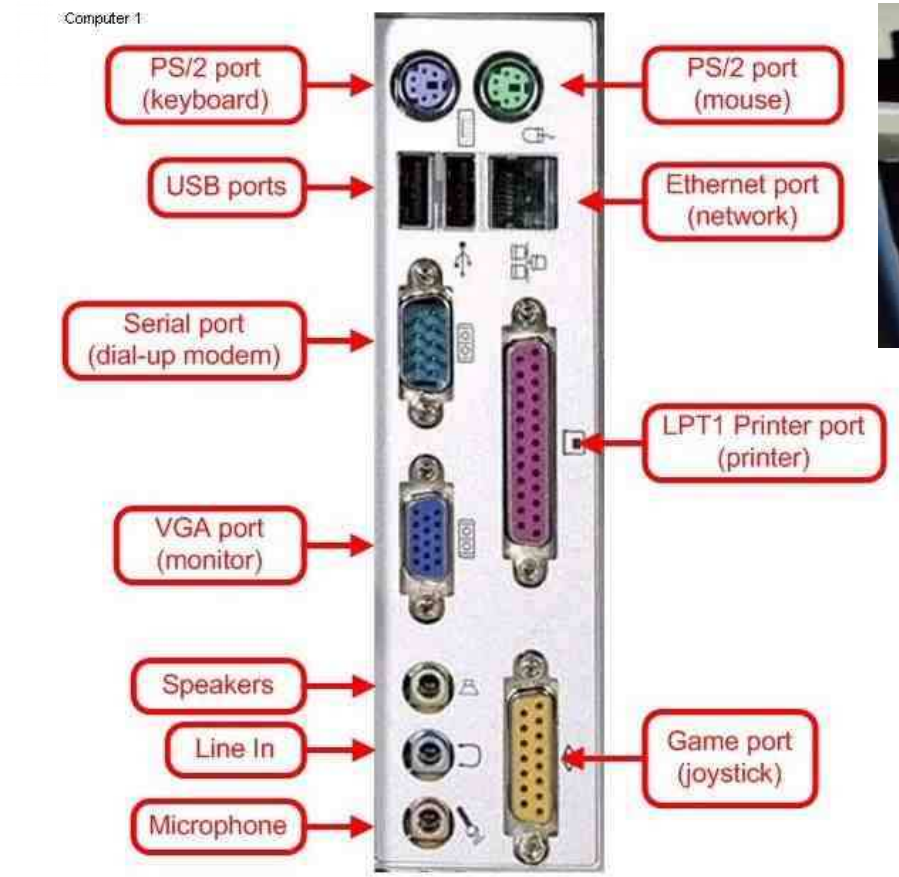


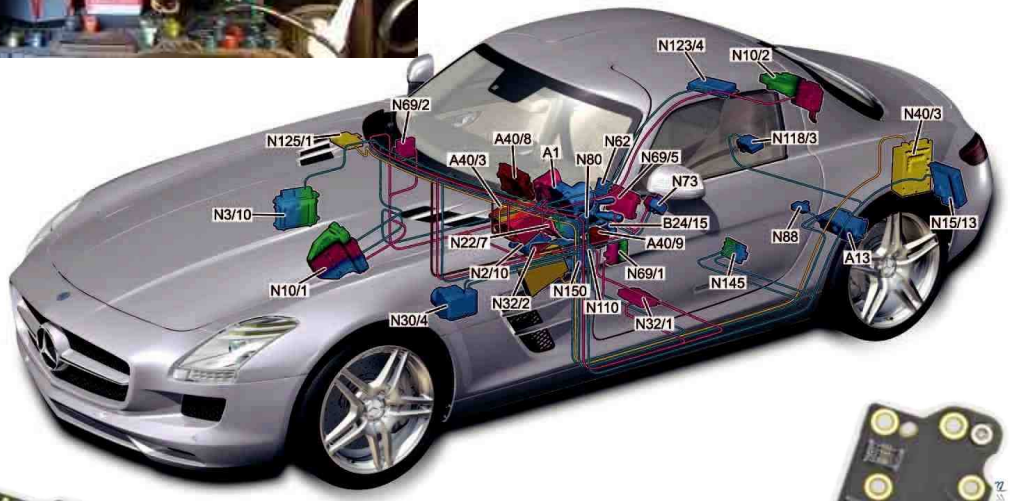
Définition d'un bus de communication

- **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.

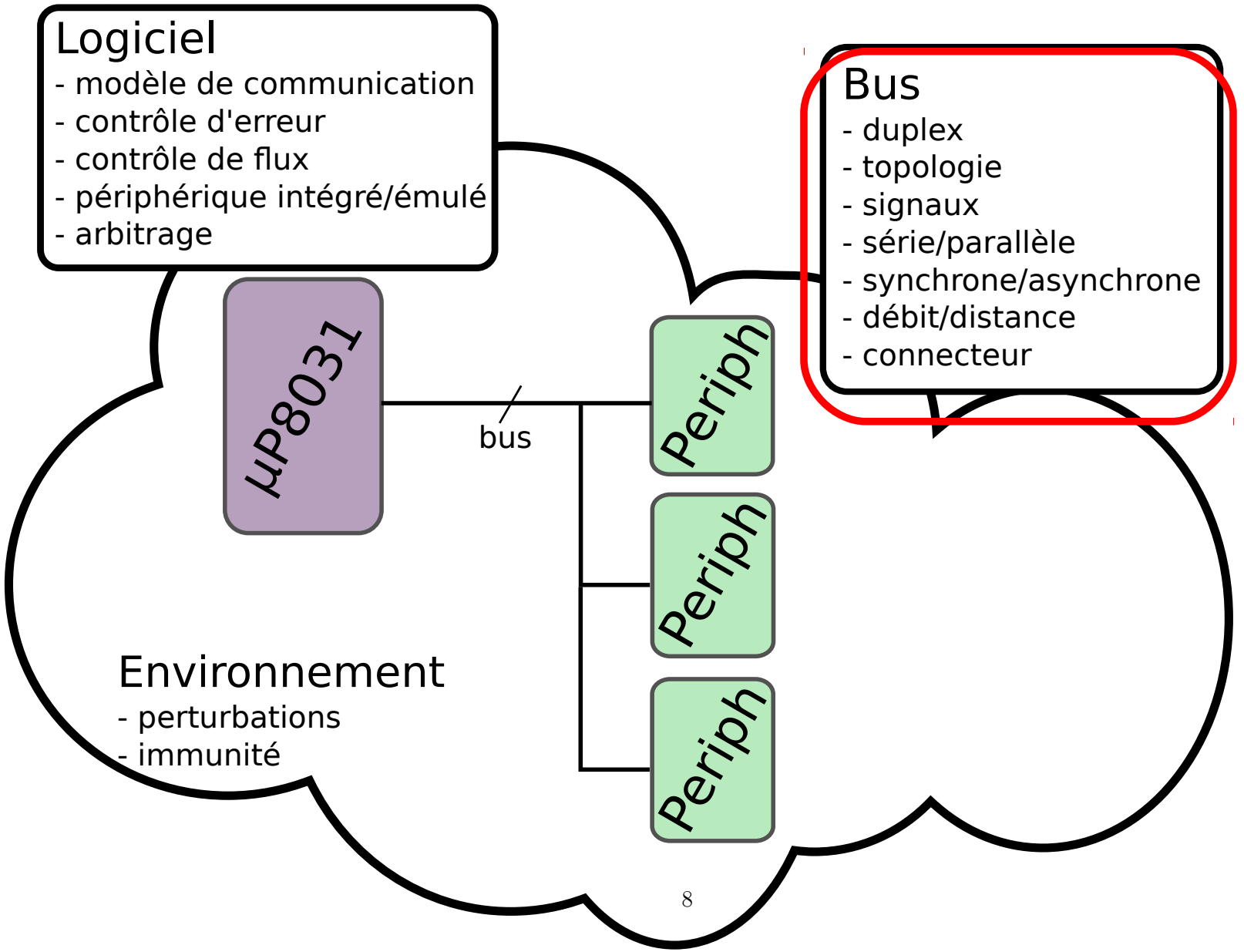
- Uniformisation par exemple communication inter-processus via Sockets UDP/TCP sur IP de boucle locale utilise API Réseau

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

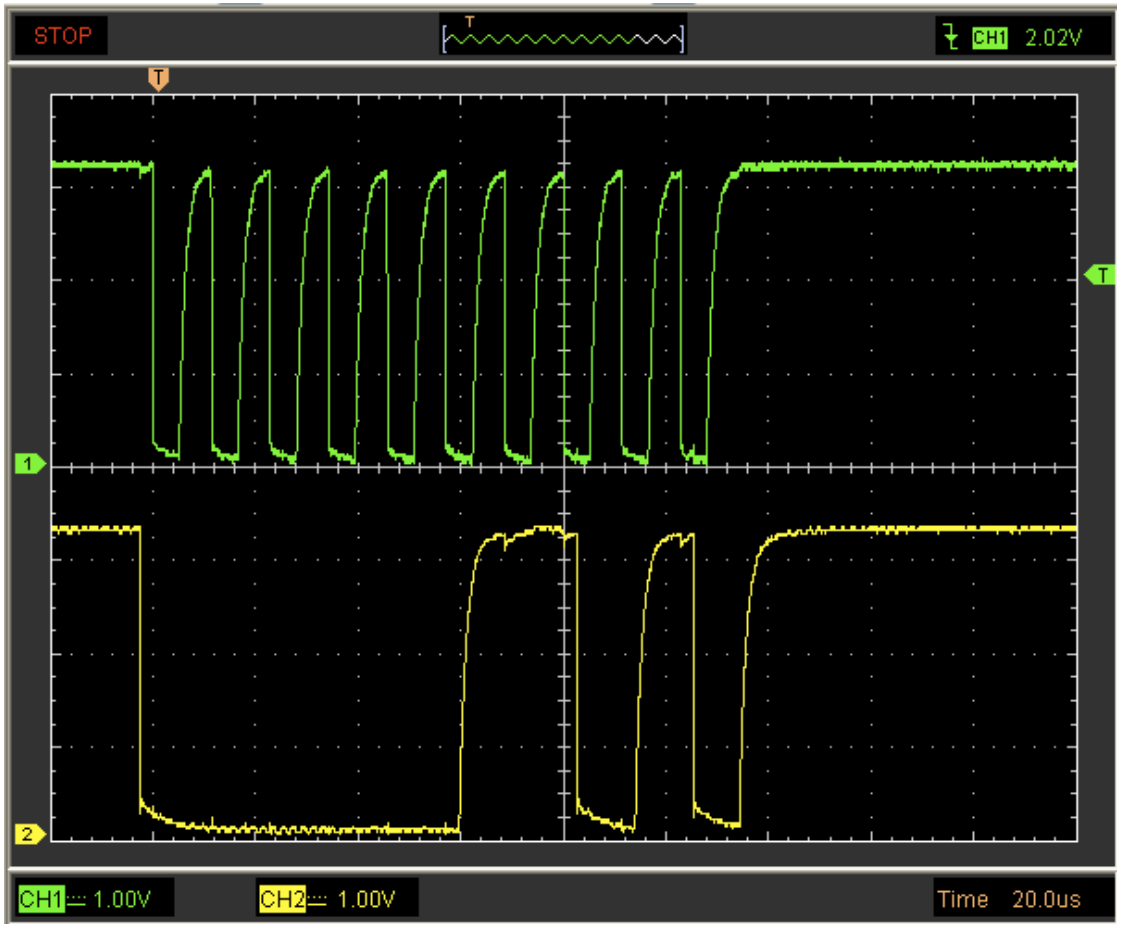
- Signaux de données:
  - Signaux véhiculant les données de la communication
- Signaux de contrôle:
  - Signaux permettant la synchronisation entre les acteurs (composants émetteurs et/ou récepteurs) du bus

Signaux

- Exemple : Transaction sur un bus I2C

Signal d’horloge

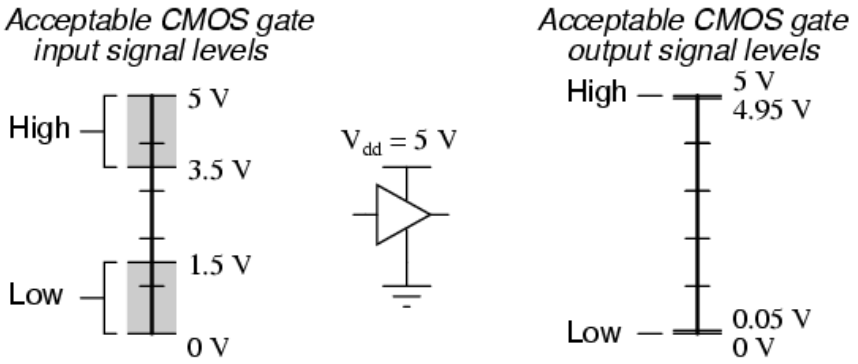
Signal de donnée



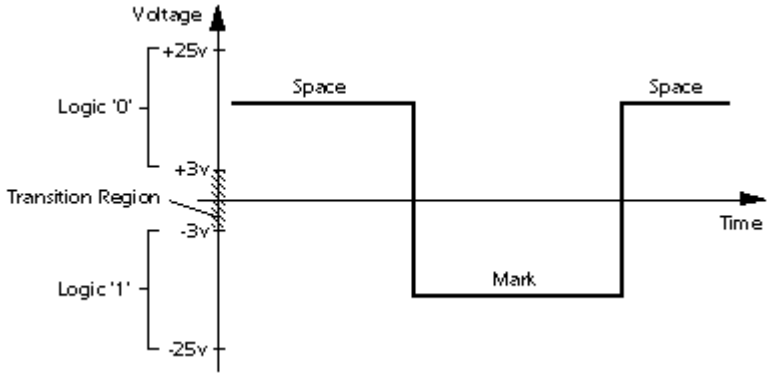
Signaux

- Caractéristiques électriques
  - Niveaux de tension, exemples :

CMOS



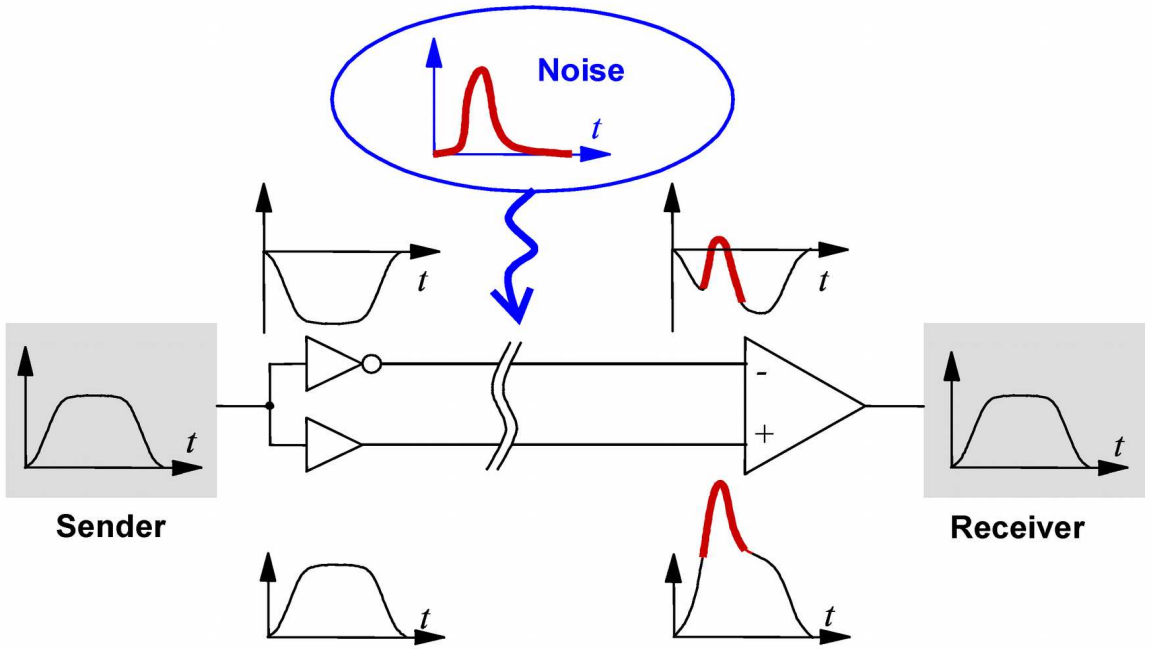
RS232

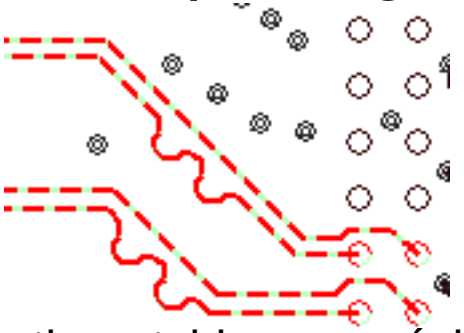


  - Courant

Signaux

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

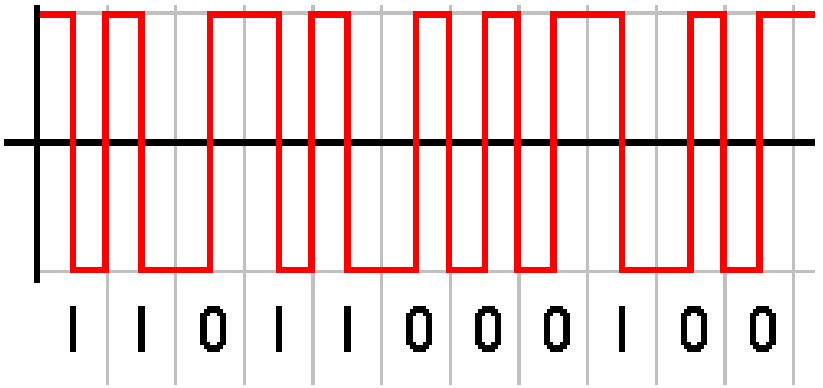




length matching pour éviter le déphasage

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



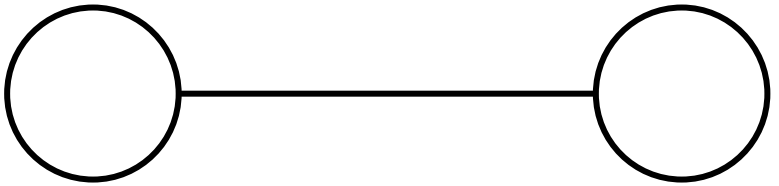
Signaux

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

- Gestion possible des conflits par 2 états :
  - Etat récessif : pouvant être altéré par un état dominant.
  - Etat dominant : ne pouvant être altéré. L'acteur imposant un état dominant, prend le contrôle du bus.

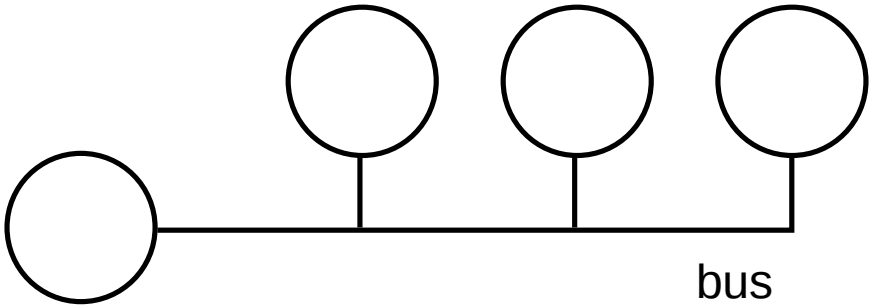
Topologie

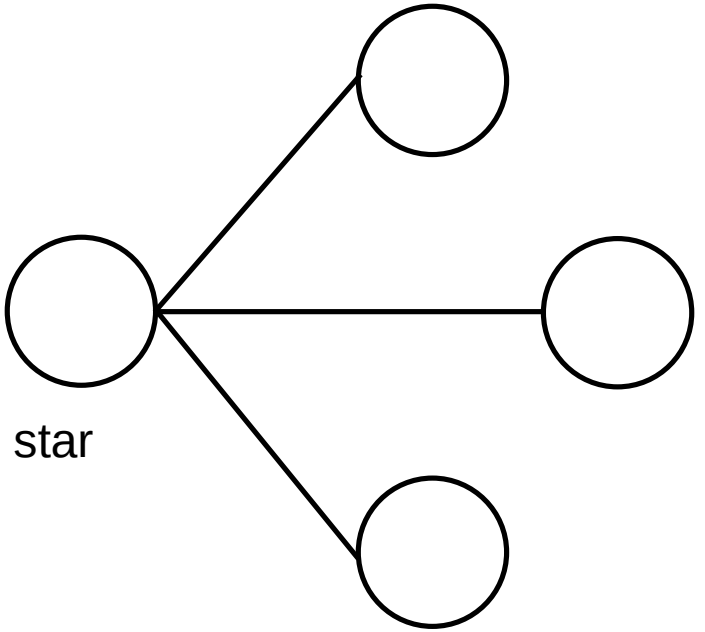
- Définit les interconnexions possibles entre les composants sur le bus
  - Un maître peut initier une communication
  - Un esclave ne peut que répondre à une sollicitation d'un maître
- Topologie Point à Point
  - Communication entre deux composants
  - Communication pair à pair ou maître-esclave



Topologie

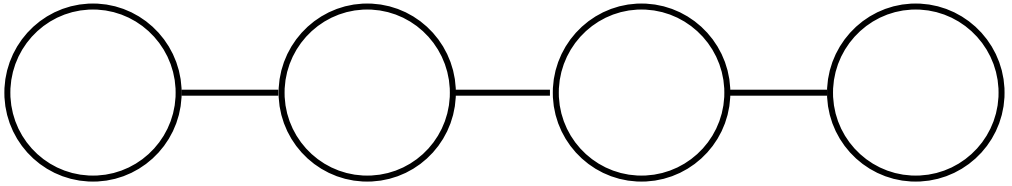
- Topologie Point → Multi-points
  - Communication entre plus de deux composants
  - Adressage
  - Communications:
    - Maître → Esclaves
    - Multi-maîtres → Esclaves
    - Pairs à Pairs





Topologie

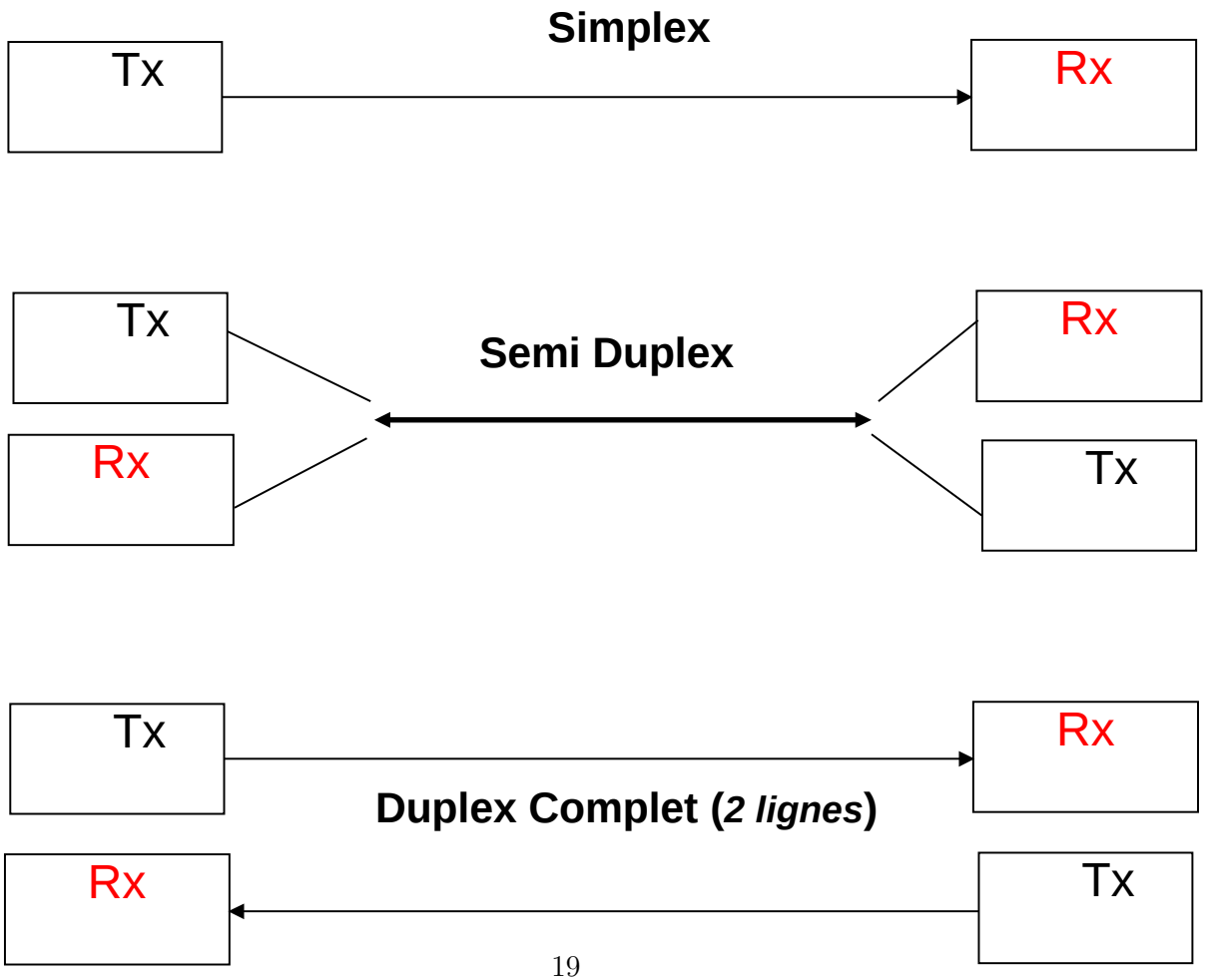
- Topologie Daisy-Chain
  - Communication entre plusieurs composants mis en cascade sur le bus
  - Chaque composant transmet la donnée à son voisin direct si elle ne lui est pas destinée



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

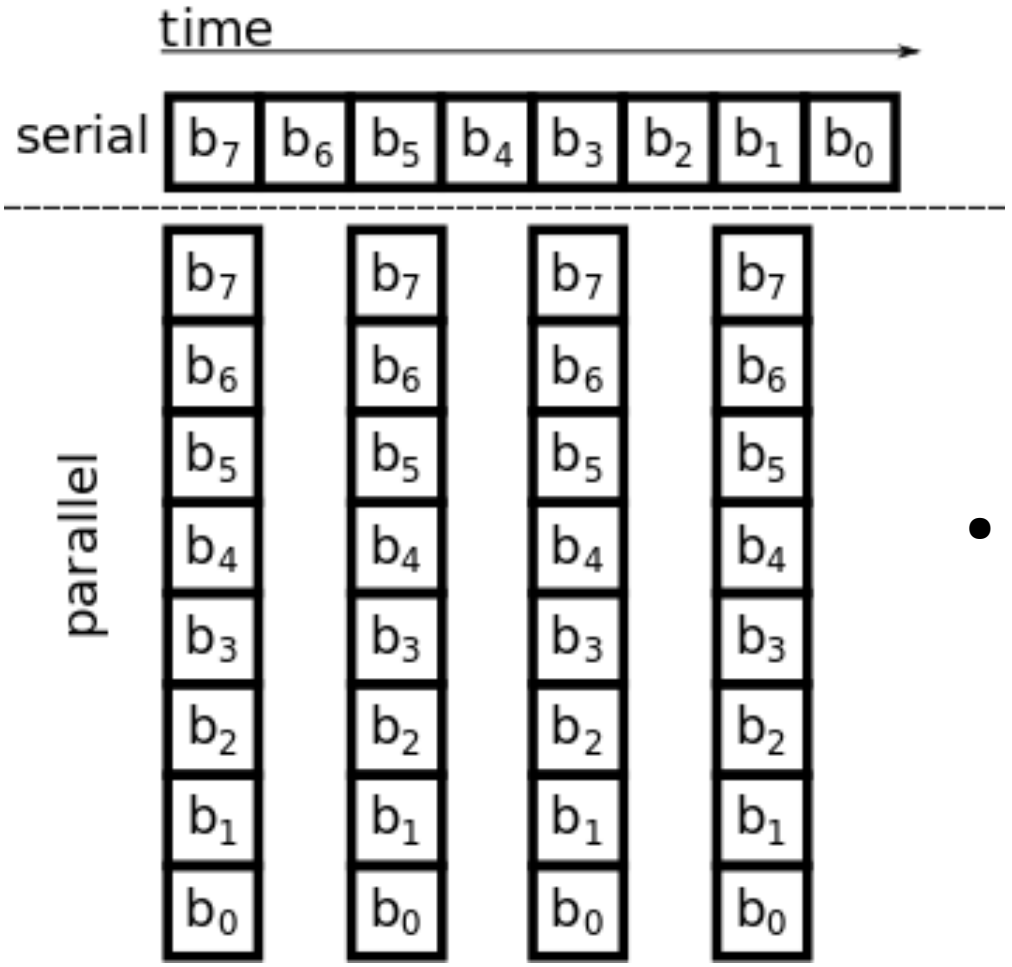


Vitesse de communication

- 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. 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 (PCIe 16X)
    - SATA vs PATA
    - Bus mémoire de processeur est parallèle
  - coût matériel de la sérialisation/désérialisation
  - latence

Bus Série Synchrone

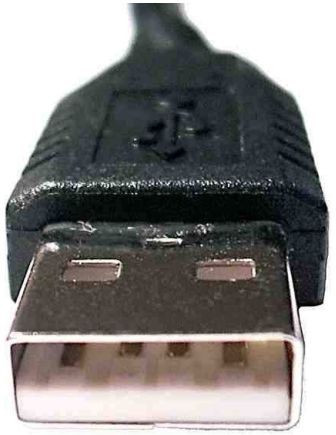
- Le bus transmet les symboles (bit) synchrones **avec une horloge**, elle aussi disponible sur le bus.
- Avantages:
  - Vitesse du bus (pas de reconstitution de la synchronisation)
- Inconvénients:
  - Câblage (nombre de conducteurs et distance)
  - Sensible au bruit (sur le signal d’horloge)

Bus 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

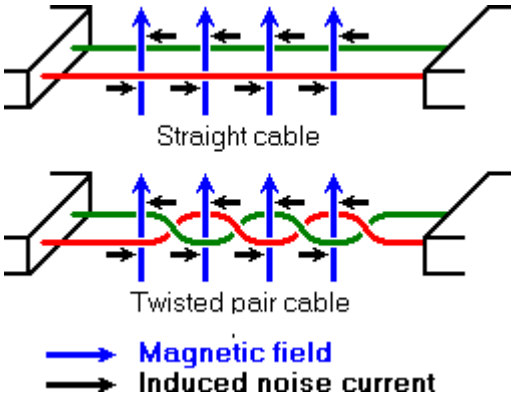


Connectique

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

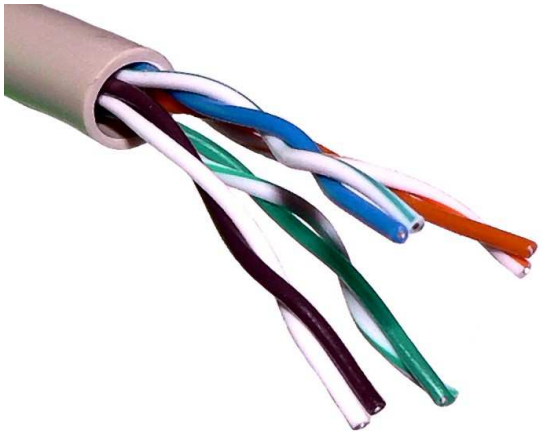
Câblage

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



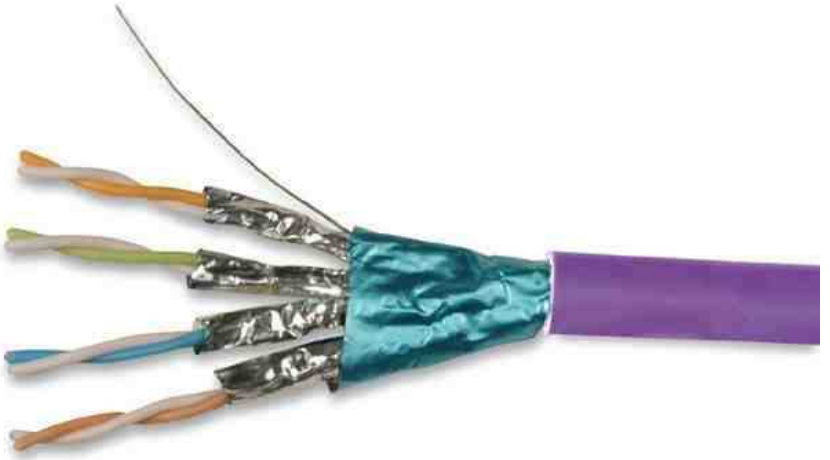
Câblage

- Exemple de cables utilisés pour réseau Ethernet



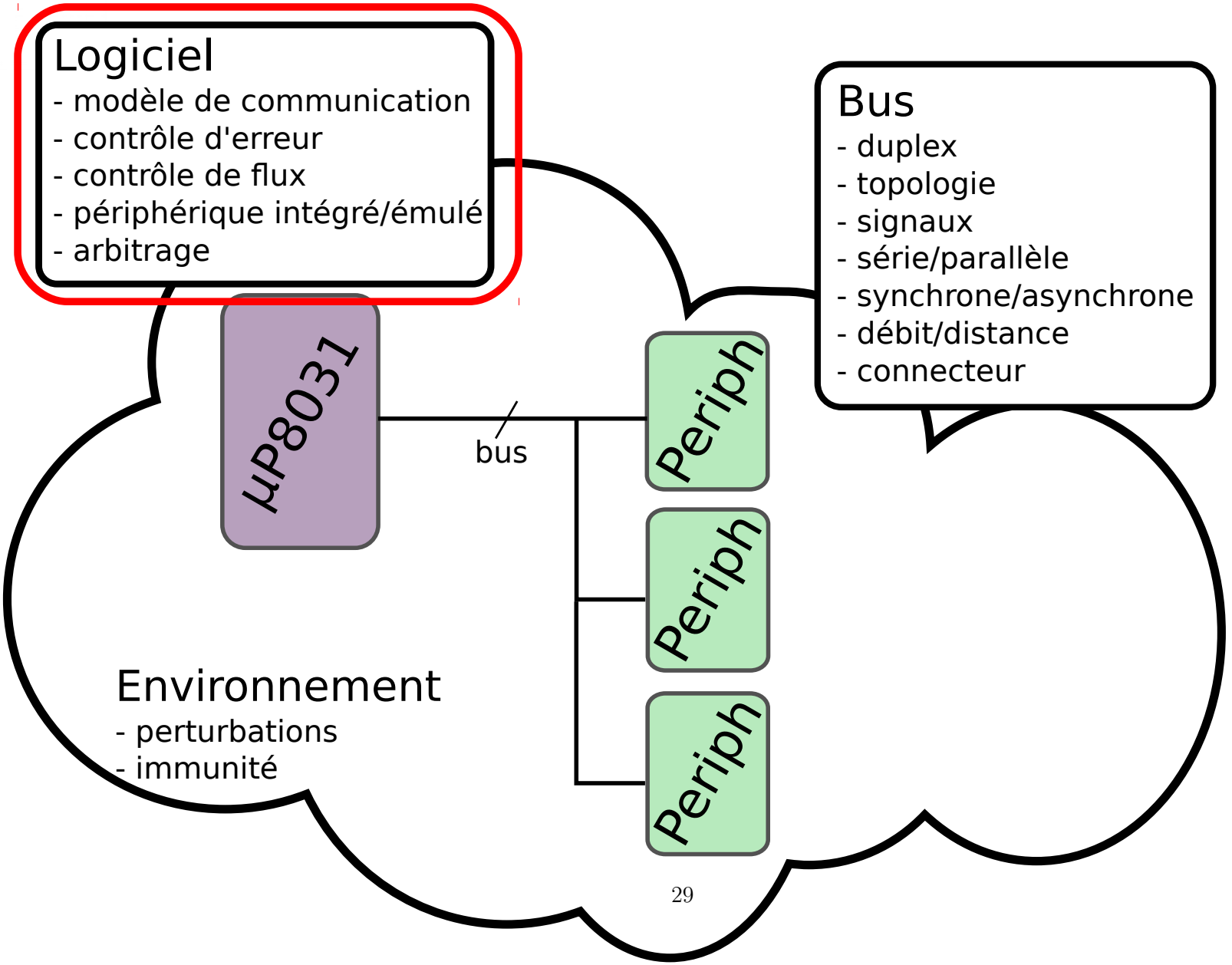
UTP
(unshielded twisted pair)

FTP (foil twisted pair): feuillard

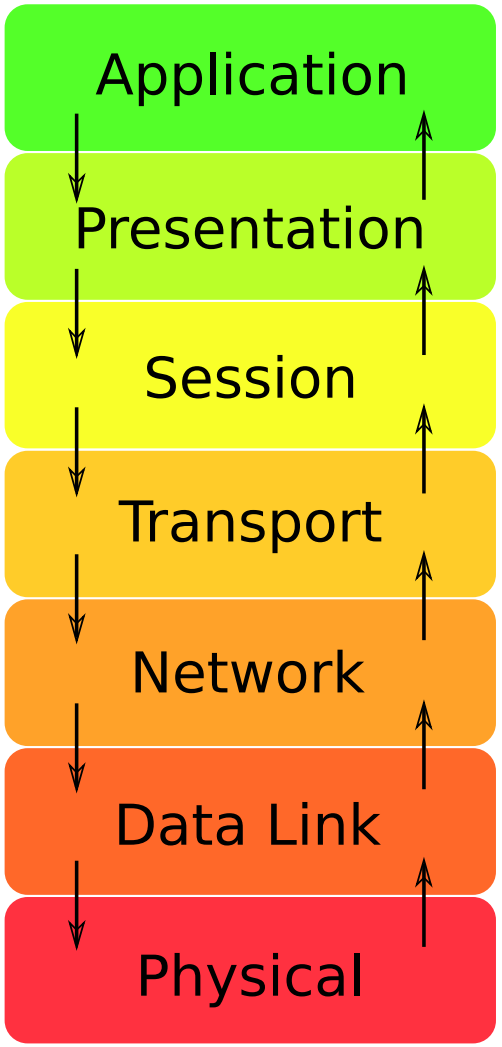


SFTP Cat7
(shielded foil twisted pair)

Sommaire

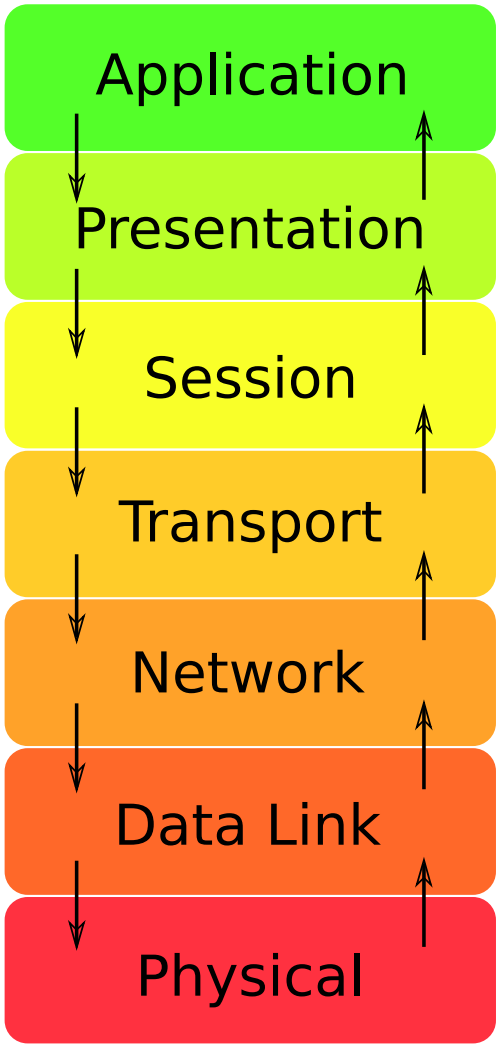

Logiciel
- modèle de communication
- contrôle d'erreur
- contrôle de flux
- périphérique intégré/émulé
- arbitrage
Bus
- duplex
- topologie
- signaux
- série/parallèle
- synchrone/asynchrone
- débit/distance
- connecteur
µP8031
bus
Periph
Periph
Periph
Environnement
- perturbations
- immunité

Couches de communication



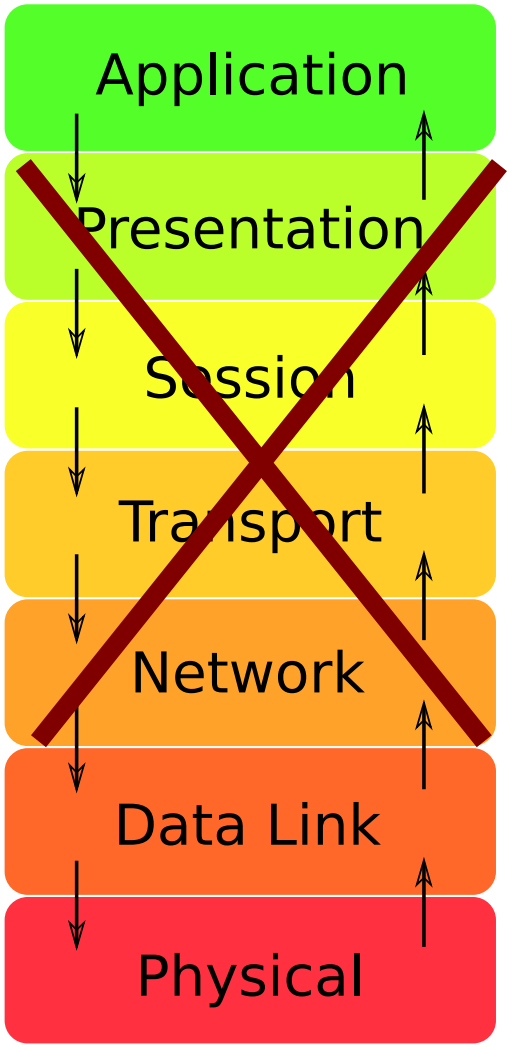
- Division en couches
- Communication entre les couches par primitives
- Unité de donnée associée à chaque couche (PDU)
- Permet la structuration des développements

Couches de communication



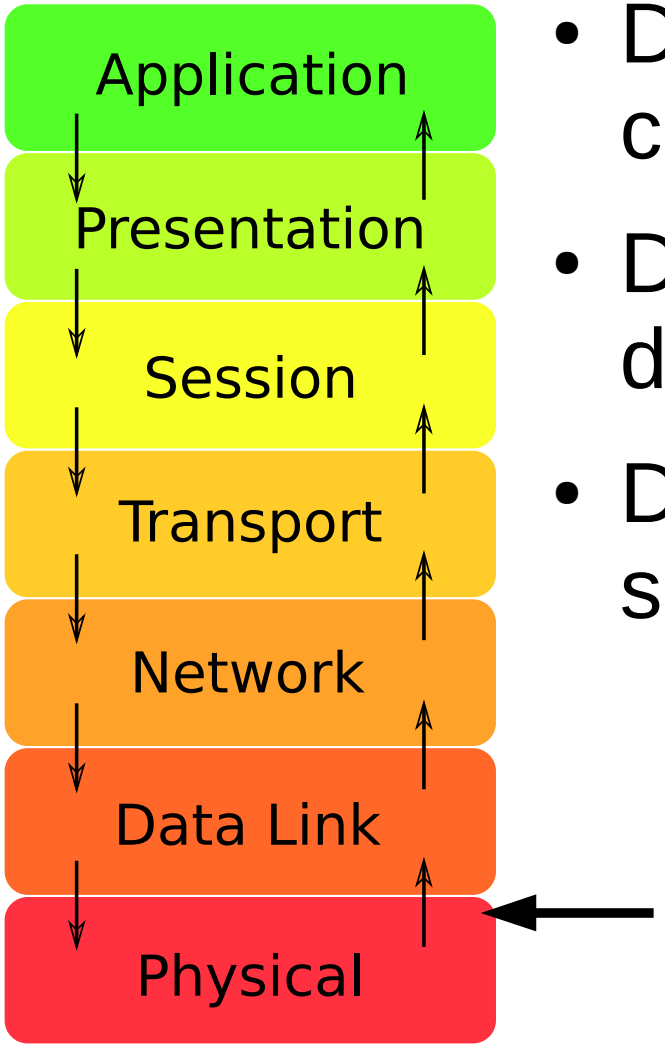
- Notion de librairie : Abstraction d'un périphérique à un certain niveau
- Ex : Librairie fournie par le fabricant d'un micro-contrôleur pour les périphériques intégrés

Couches de communication



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

Couche Physique

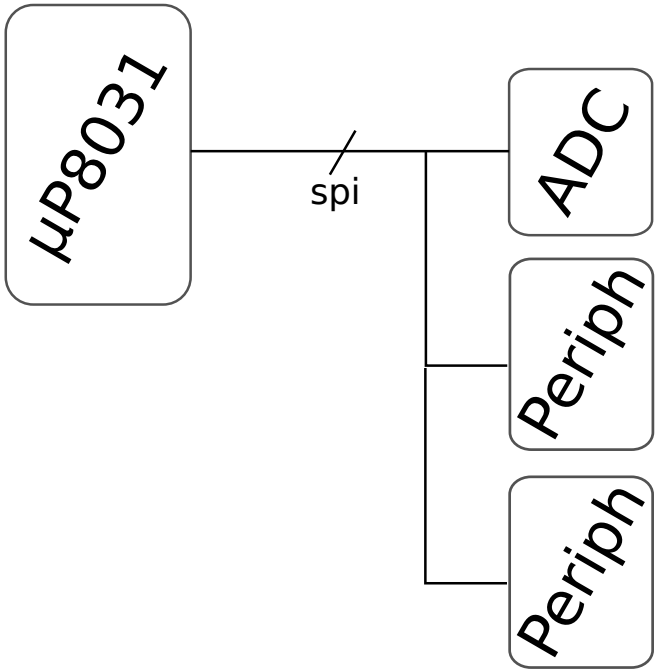


- Définit l'interface de communication sur le média
- Définit le codage des bits de donnée sur le bus
- Définit les topologies de bus supportées

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



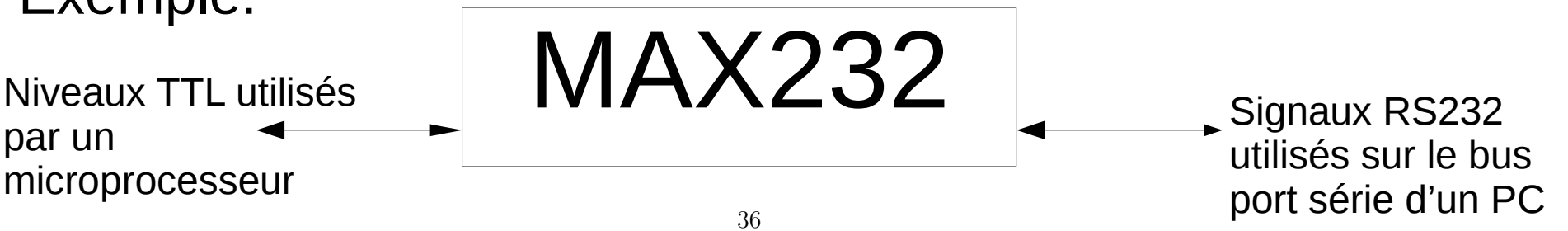
Physical

uchar transfer_byte_spi(uchar send)

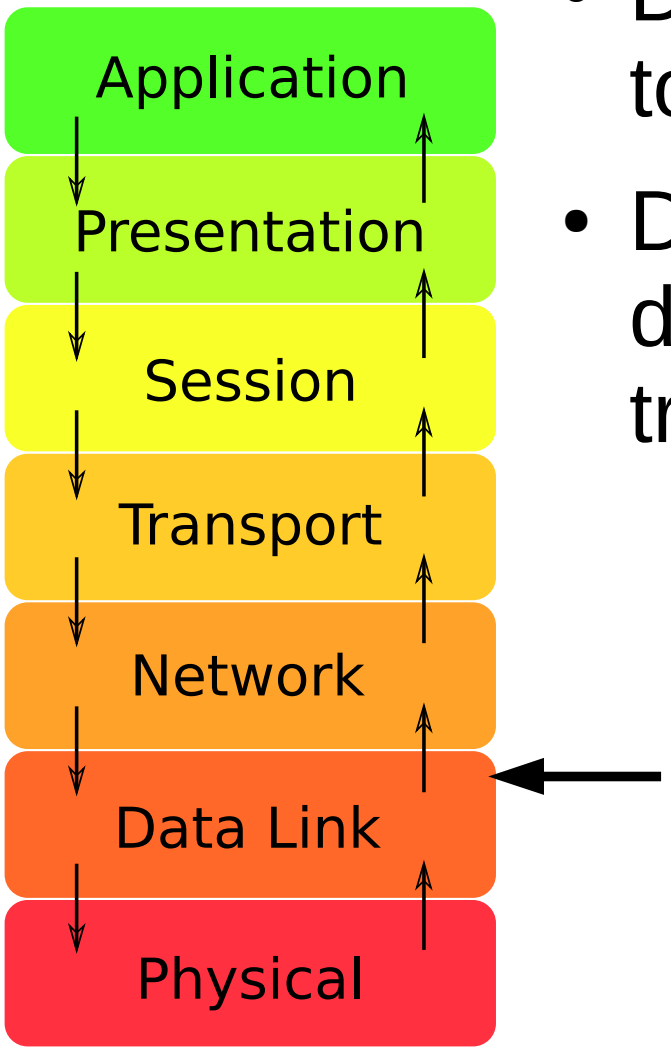
Couche Physique

- 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

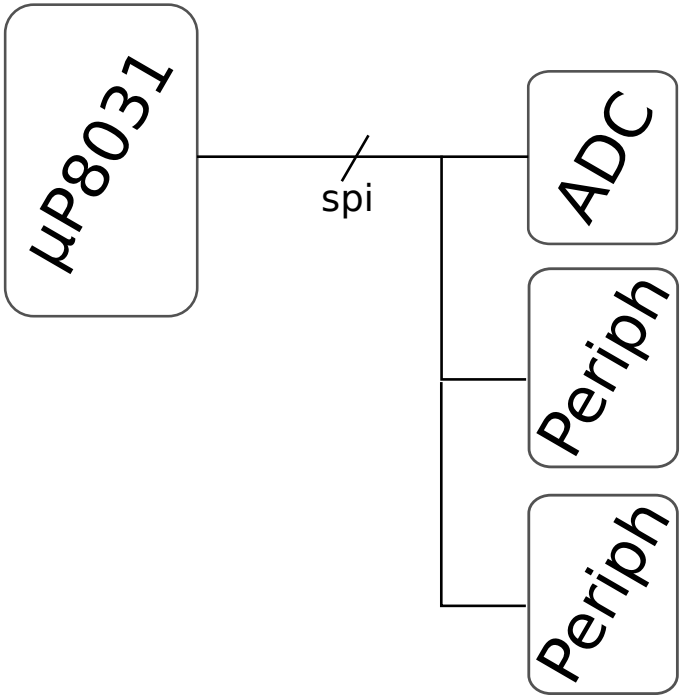


- Définit l'adressage dans le cas des topologies point → multi-points
- Définit le formatage et le découpage des données transmises sur le bus

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)

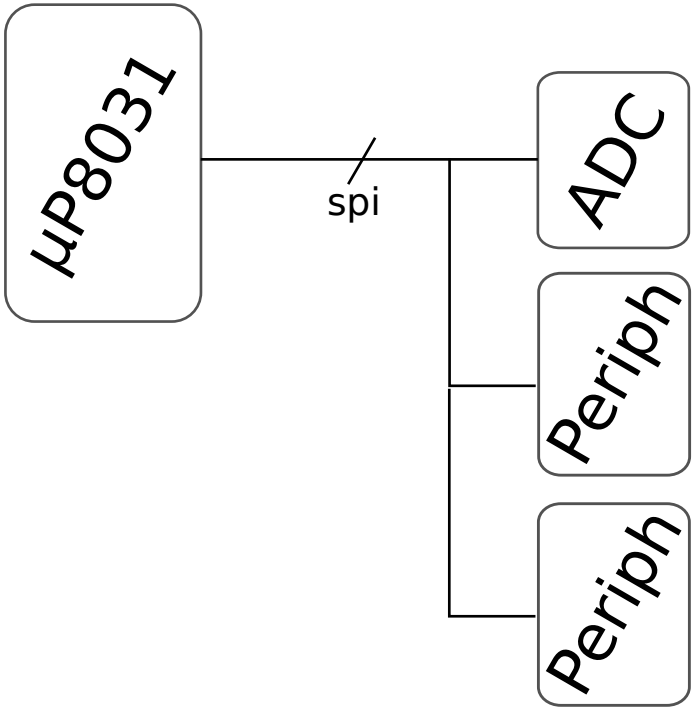
Couche Liaison



Data Link — uchar transfer_buffer_spi(uchar addr, uchar * send, uchar * rcv, uint nb)

Physical — uchar transfer_byte_spi(uchar send)

Couche Application



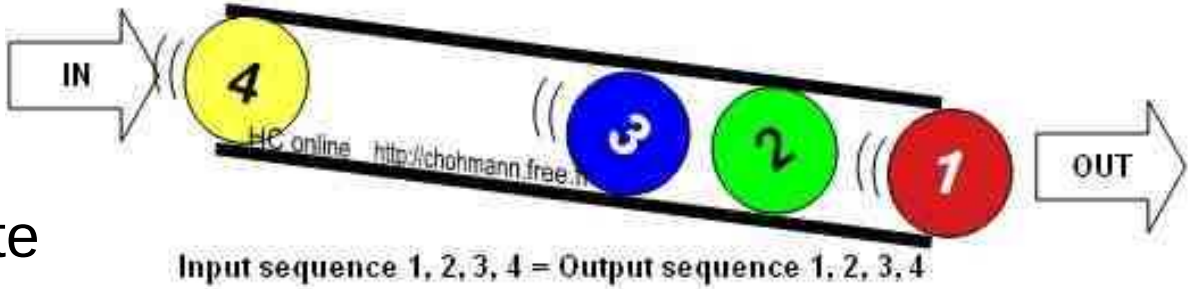
Application	readADCChannel(uchar nb, uint * val)
Data Link	uchar transfer_buffer_spi(uchar addr, uchar * send, uchar * rcv, uint nb)
Physical	uchar transfer_byte_spi(uchar send)

Modèle de communication

- Transaction de mémoire
  - Données organisées en “paquets”
  - Besoin de synchroniser

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 token (jetons) dans la FIFO
  - Le consommateur lit des tokens dans la FIFO
  - La lecture/écriture préserve la séquentialité des données
  - Token = 1*uchar dans les cas qui nous intéressent



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 distance ;

};


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

Modèle de communication

```
bit fifo_read(struct my_fifo *  fif, unsigned char * data) {
        if(fif->distance == 0) return 0 ;
        *data = fif->buffer[fif->read_index];
        fif->read_index = (fif->read_index + 1) % FIFO_SIZE;
        fif->distance--;
        return 1;
}
bit fifo_write(struct my_fifo * fif, const unsigned char c) {
        if (fif->distance <  FIFO_SIZE) {
                fif->buffer[fif->write_index] = c;
                fif->write_index = (fif->write_index + 1) % FIFO_SIZE;
                fif->distance ++;
                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.
  - 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
- 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	000000000	000000001
10100001	3	101000011	101000010
11010001	4	110100010	110100011
11111111	8	111111110	111111111

Contrôle d’erreur, CRC, Checksum...

- Utilisation d'un jeux d'octets réduit
  - Codage des données en ASCII, permet de faciliter la détection d'erreurs
  - Si réception d'un caractère non-ASCII, erreur ...
- Somme de contrôle :
  - clé permettant de vérifier l'intégrité des données
  - calculée par la somme ou le 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

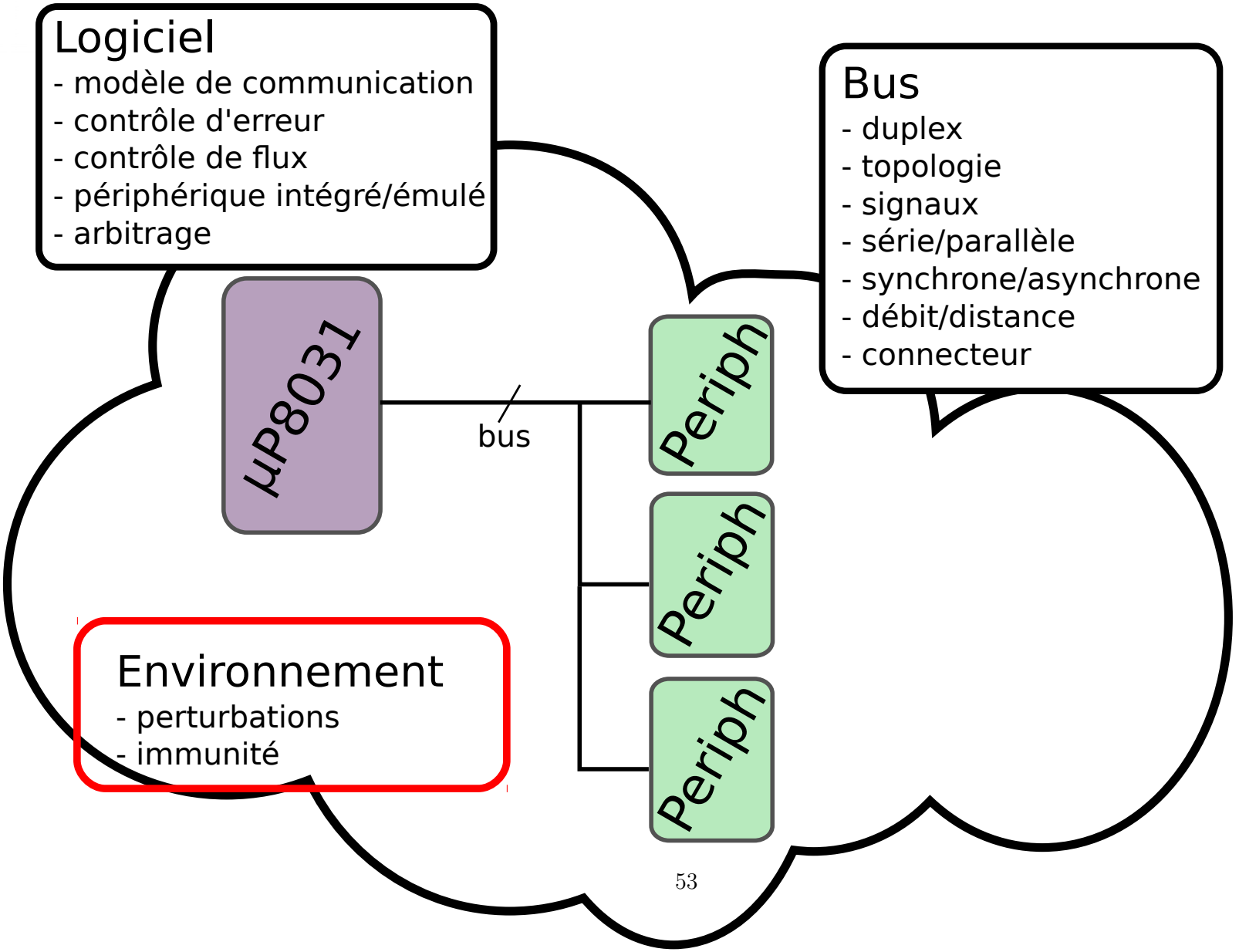
Périphérique Intégré (Matériel)

- 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 le 8250

Périphérique Emulé (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
- 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

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

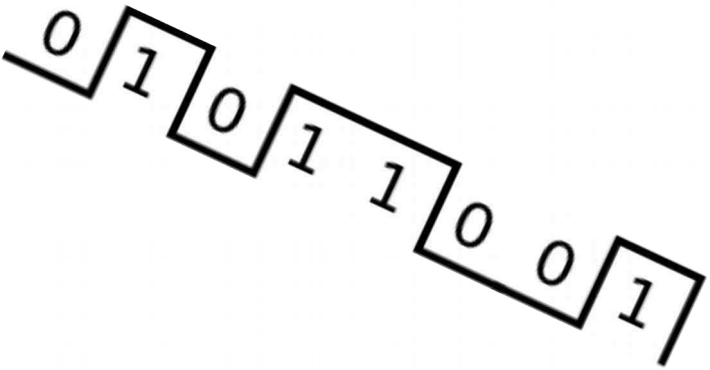
- Cross-Talk (diaphonie par induction électromagnétique)
- Rayonnement électromagnétique externe
  - Alimentation alternative
  - Machine-outils
  - Réseaux sans fils
- Longueur des lignes
- Impédance des lignes et terminaisons

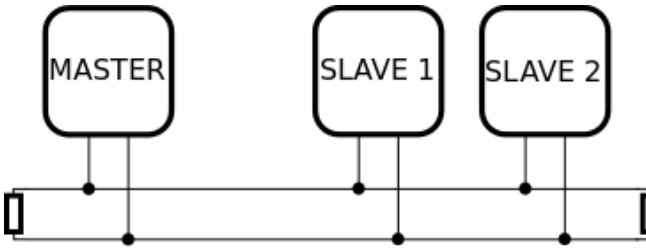
Immunité

- 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



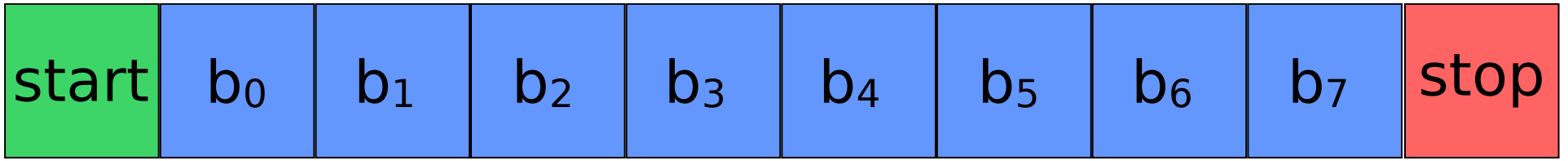


Bus Série asynchrone

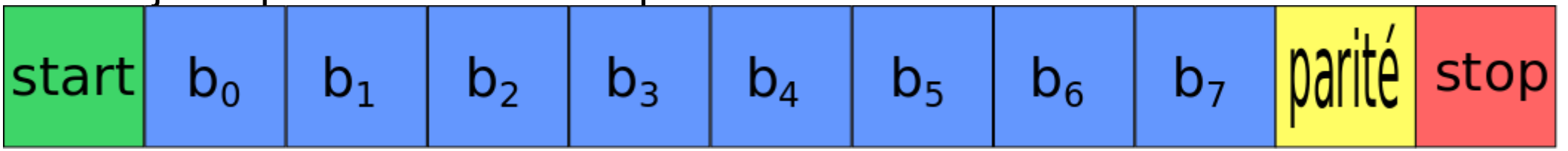
- Le bus ne véhicule pas de signaux de synchronisation
- La synchronisation est récupérée par les données
- Avantages :
  - Moins de conducteurs pour former le bus.
  - Moins sensible au bruit.
- Inconvénients :
  - La reconstitution de l'horloge à partir des données nécessite un traitement
  - Vitesse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

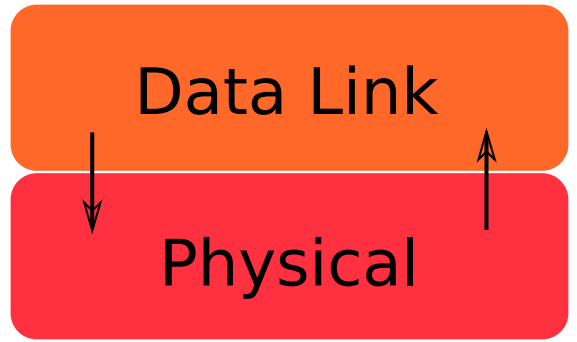
- Caractérisation, (triplet)
  - Un chiffre indiquant le nombre de bits de donnée utiles :
    - (5, 6, 7, 8, 9, 10 …)
  - Une lettre indiquant le type de parité utilisé :
    - N pas de parité
    - E parité paire (Even)
    - O parité impaire (Odd)
  - Un chiffre indiquant le nombre de bits de stop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

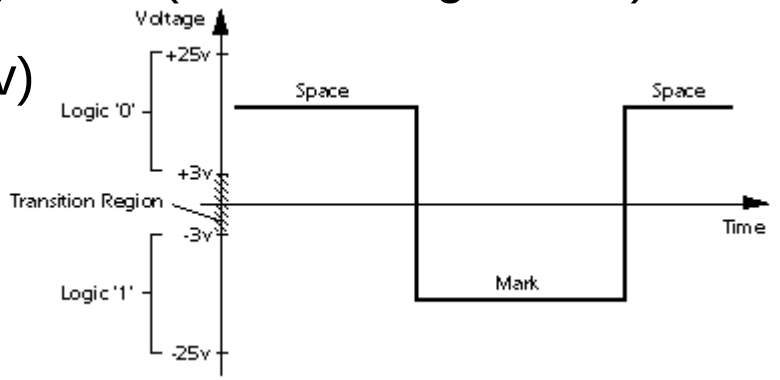
Liaison UART

- Baud-rate : nombre de symboles transmis par seconde
  - Pour liaison UART, Baudrate = Bitrate, nombre de bits transmis par secondes
- 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+start+1*stop+0*parité) → (8/10)*baudrate = 80% du baudrate
    - 8E2 → 8/(8+start+2*stop+1*parité) → (8/12)*baudrate = 66% du baudrate

RS232, Carte d'identité

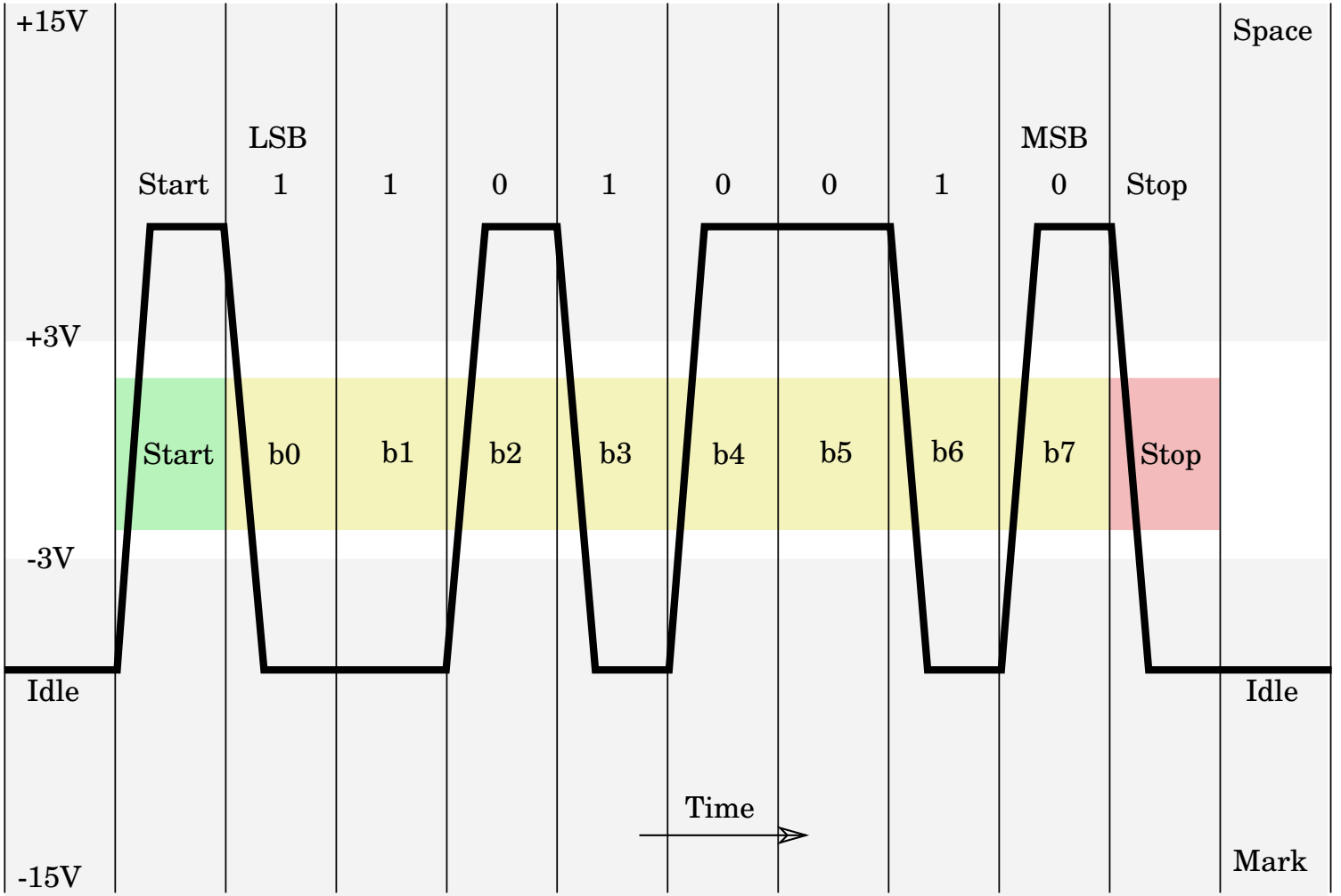
- Bus série asynchrone
- Topologies supportées:
  - Point → Point (en pair à pair)
- Full-duplex (ou simplex)
- Spécification électriques:
  - '0' logique codé par tension positive (3v < Vlow < 25v)
  - '1' logique codé par tension négative (-25v < Vhigh < -3v)
  - zone non définie (-3v < Vidle < 3v)





RS232, Exemple de trame

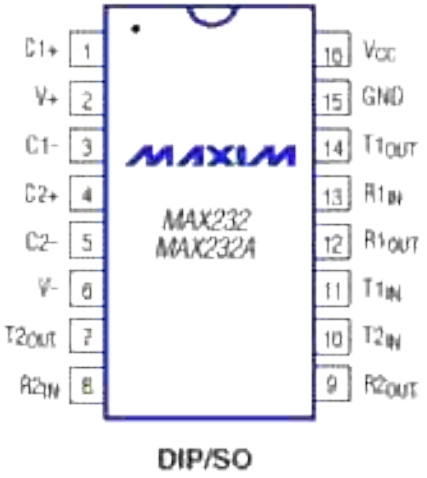
Transmission 8N1



Source : Wikipedia

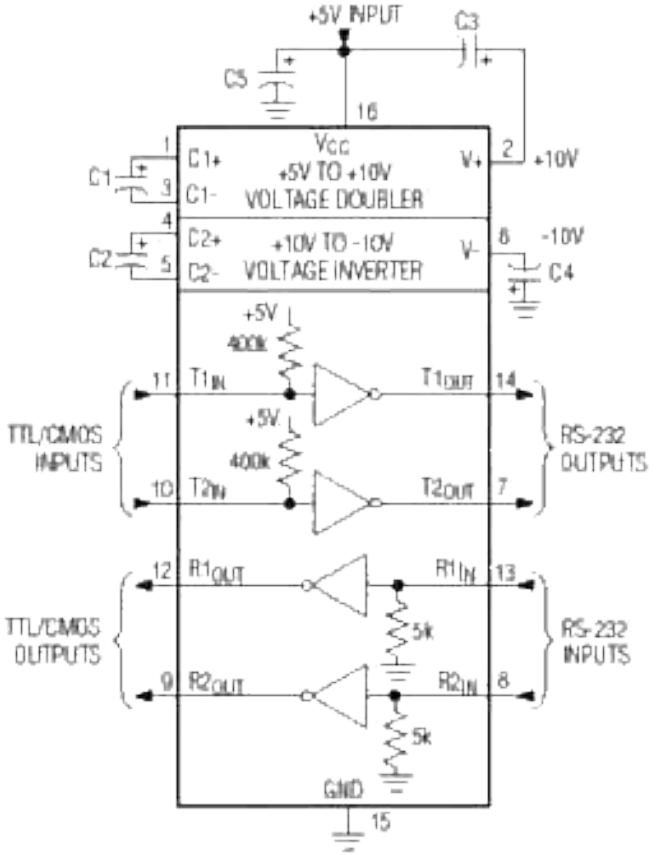
RS232, PHY

- Composant PHY
  - Ex : Rs232 TTL → RS232 : MAX232
  - Génération des tensions utilisées par le signaux réalisée par le composant grâce à circuit à pompe de charge



DIP/SO

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



RS232, Cablage

- 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)

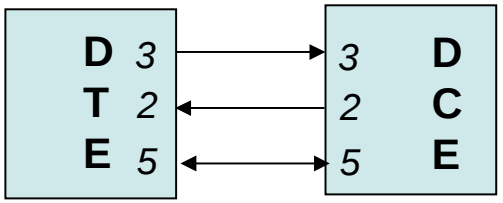
RS232, Exemple de cablage

2 types d'équipements :

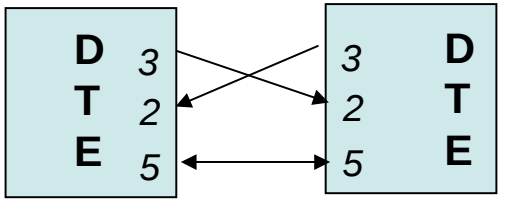
DTE *(Data Terminal Équipement) : Terminaux et PCs qui transmettent et reçoivent les données.*

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

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

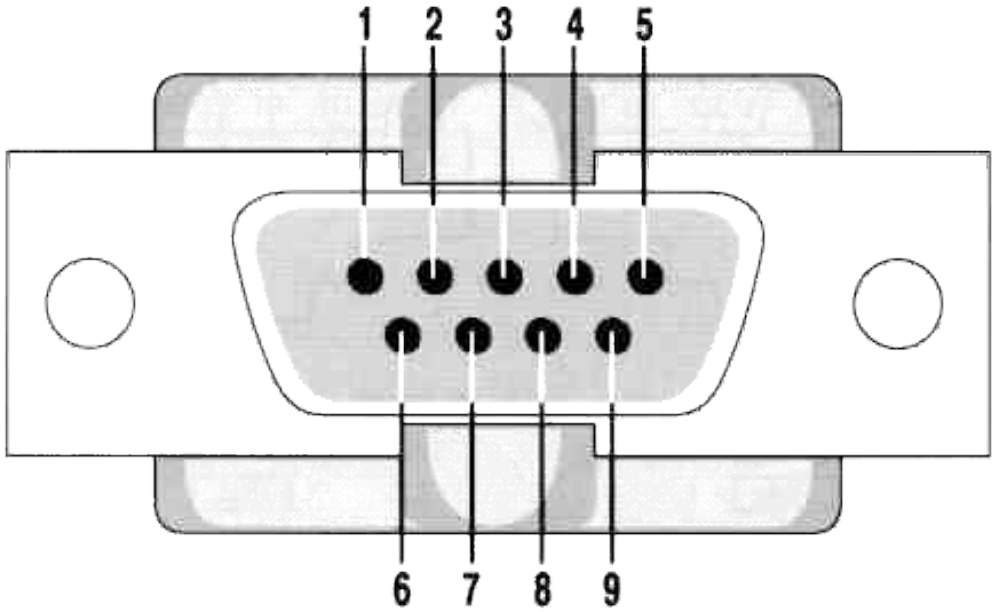


Connexion PC-Modem avec connecteurs DE9 par cable DROIT



Connexion PC-PC avec connecteurs DE9 par cable CROISE

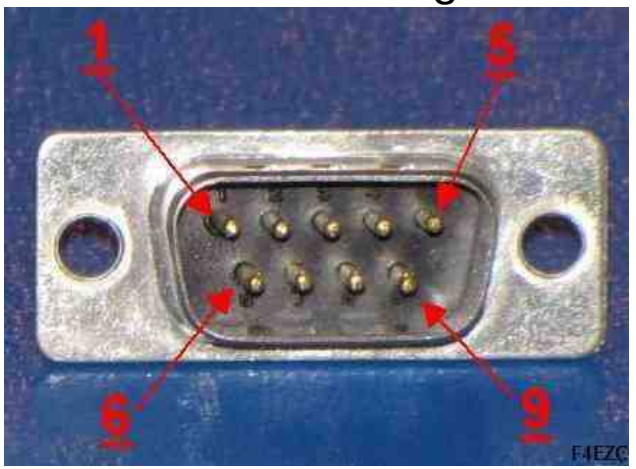
RS232 brochage DTE



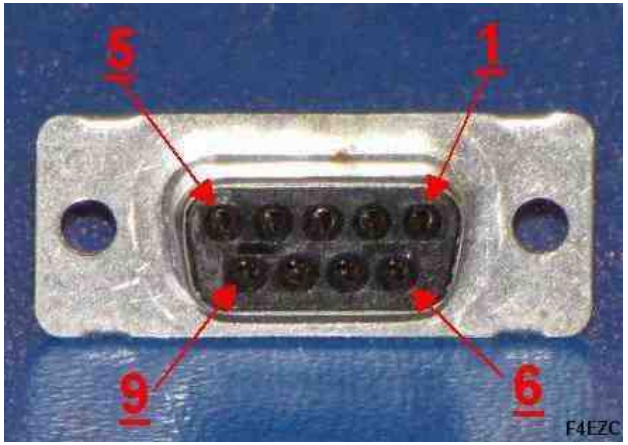
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Source: www.arcelect.com

Numérotation des broches différente selon le genre



Sur connecteur mâle

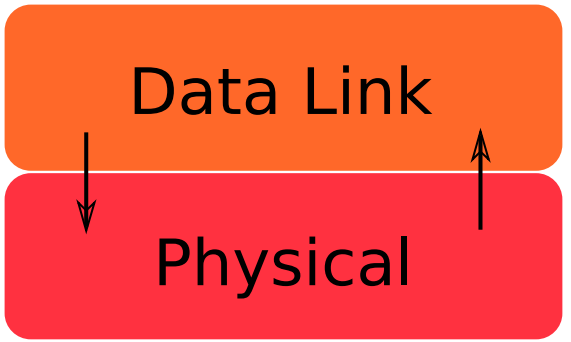


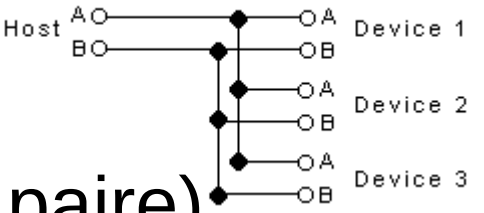
Sur connecteur femelle

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 (32 max)
    (en maître-esclave)
- Full-duplex (2 paires), Half-duplex (1 paire)
- Spécification électriques:
  - Transmission différentielle
  - +5v, -5v tension différentielle
  - 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

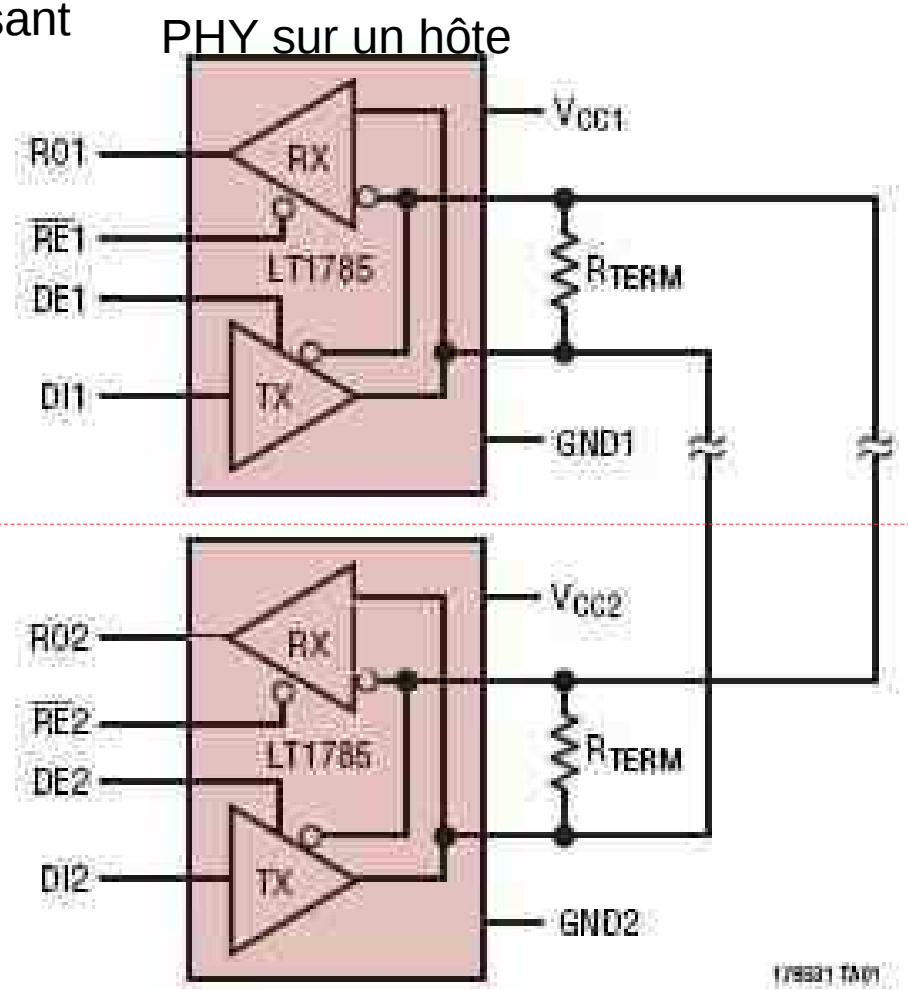
signaux au niveau du composant par exemple TTL

Signal reçu

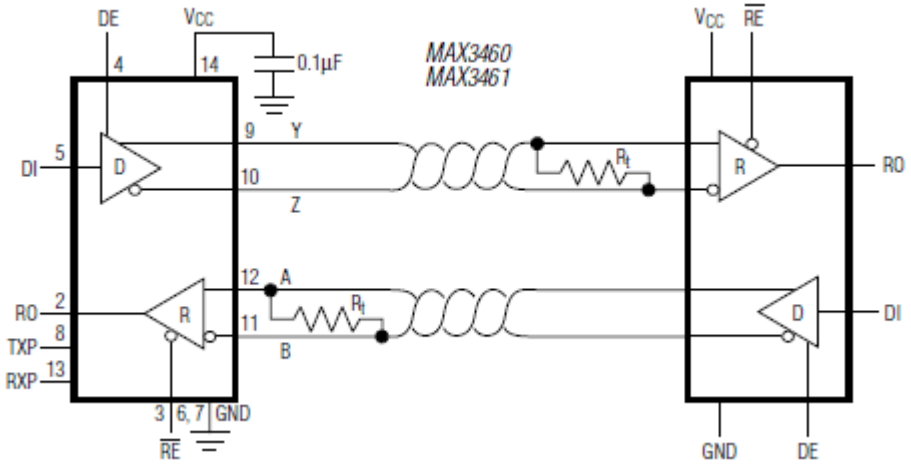
Autorisation de réception

Demande d'émission

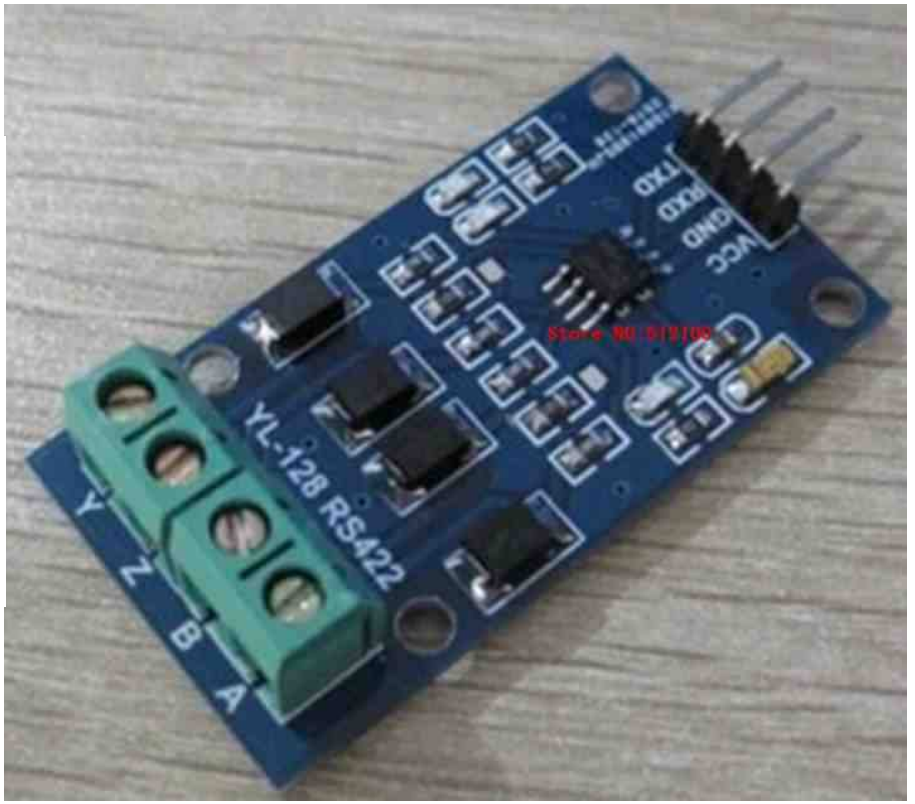
Signal à émettre



RS422, PHY



Connexion de deux PHY
à l'aide de deux paires torsadées

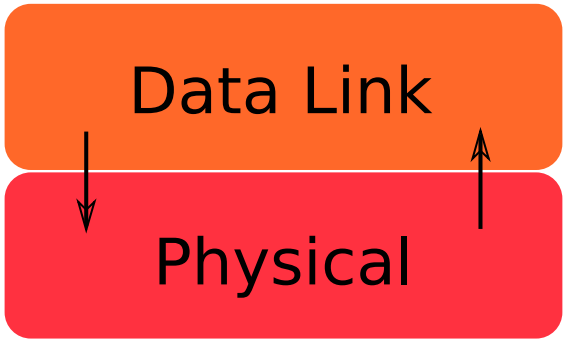


Module d'interface TTL<→RS422

RS485, Carte d'identité

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

    (en maître/esclaves)

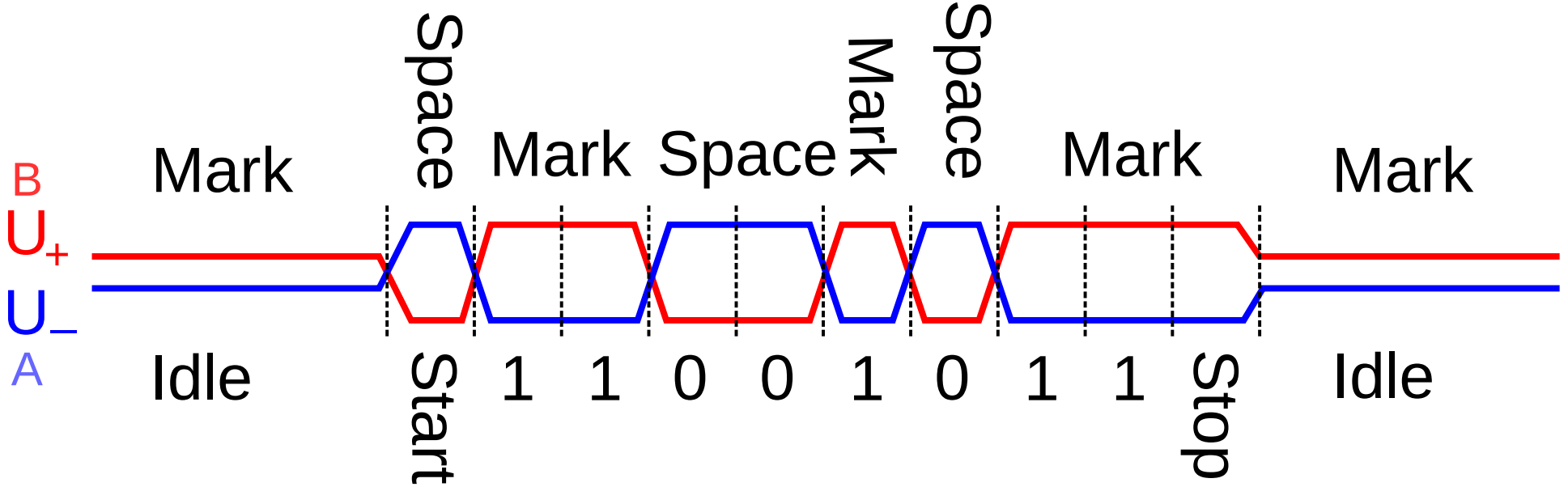


- Full-duplex (2 paires), Half-duplex (1 paire)
- Spécification électriques:
  - Transmission différentielle
  - +200mv, -200mv tension différentielle
  - Valeur du bit encodée par le signe de la tension différentielle

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 sur RS-232
- 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)

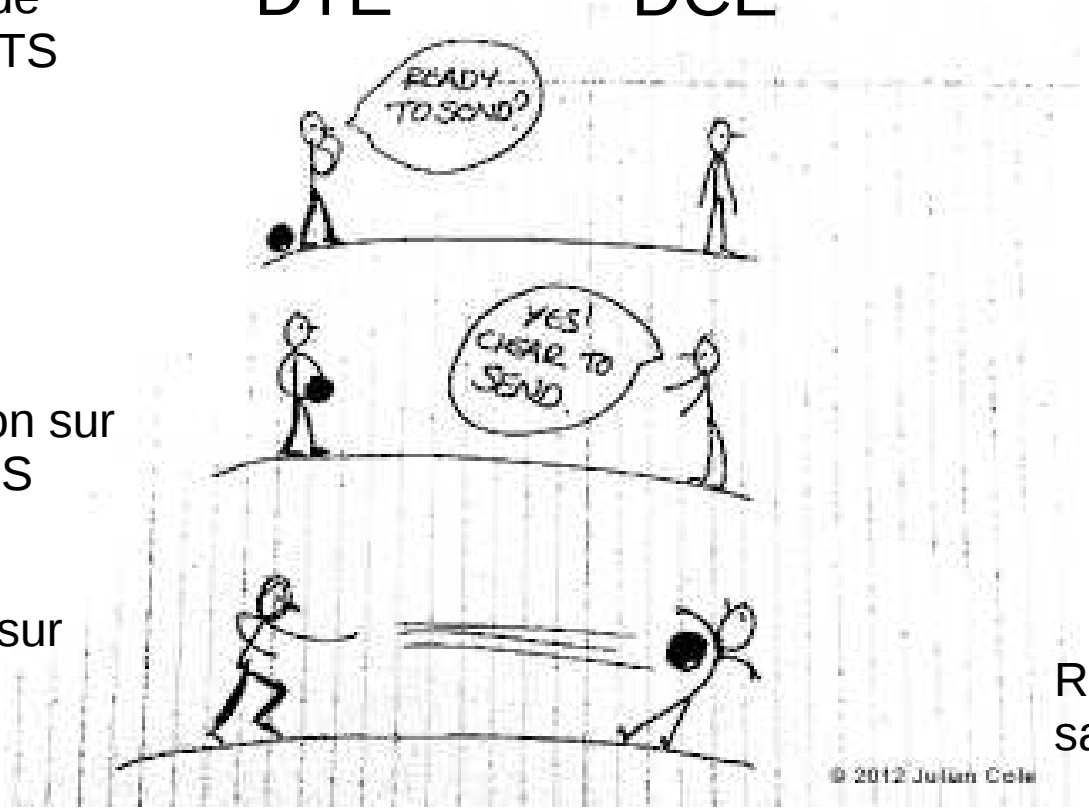
Contrôle de flux **matériel**

- Procédure RS232 RTS/CTS :

Le DTE souhaite transmettre des données au DTE

DTE DCE

Emet la demande sur sa broche RTS (sortie)



Reçoit la demande sur sa broche CTS

Emet l'autorisation sur sa broche RTS

Reçoit l'autorisation sur sa broche CTS (entrée)

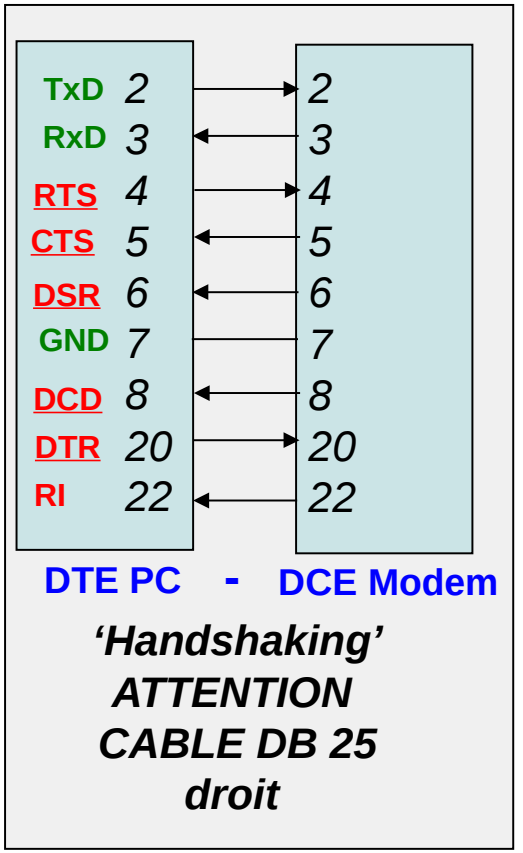
Emet les données sur sa broche TX

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.

Contrôle de flux **Logiciel**

- Utilisation d'une solution logicielle : Utilisation de caractères spéciaux
    - XON (11h) et XOFF (13h)...
- Ne nécessite pas de signaux supplémentaires
- Héritage des imprimantes texte
- Contrôle de flux asymétrique
- Utilisable en mode texte uniquement

Contrôle de flux **Logiciel**

- Procédure
  - Receveur envoie XON
  - Emetteur débute l'envoie des données
  - Receveur envoie XOFF dès qu'il ne peut plus recevoir de données
  - Emetteur stoppe l'envoie des données
  - Receveur envoie XON dès qu'il peut de nouveau recevoir des données
- 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.

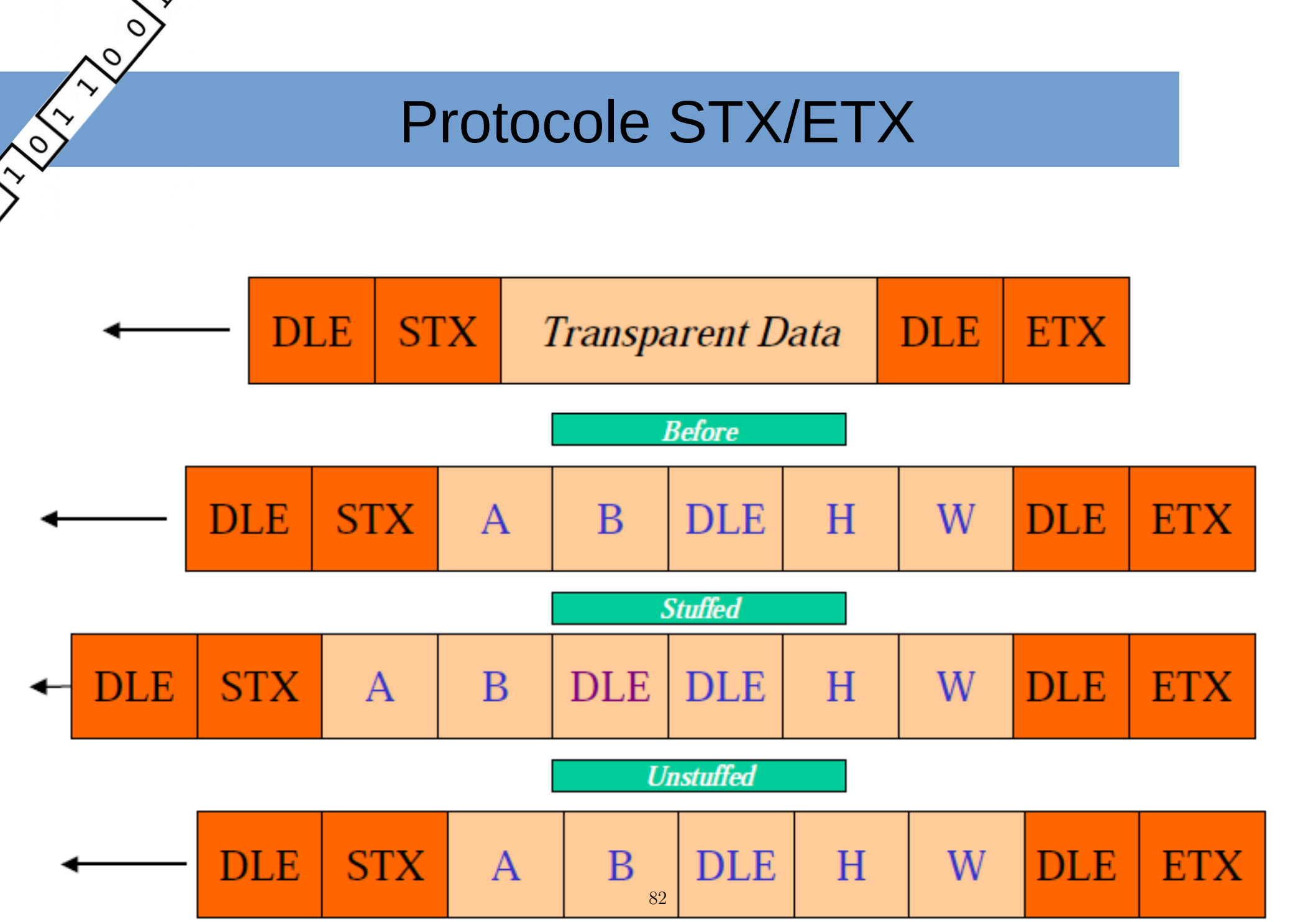
UART et couche réseaux

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

Protocole STX/ETX

- Protocole de transmission synchrone 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

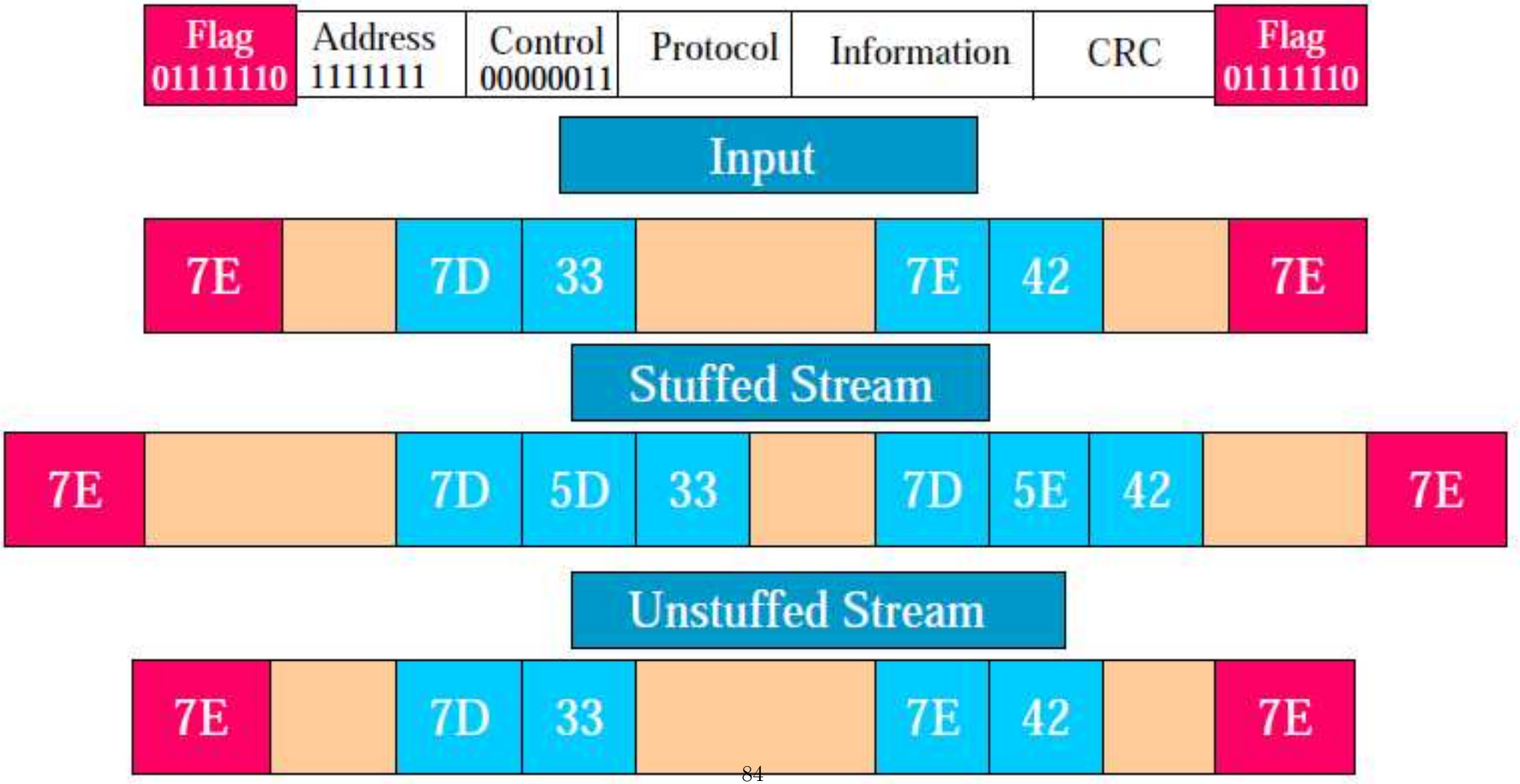
Protocole STX/ETX



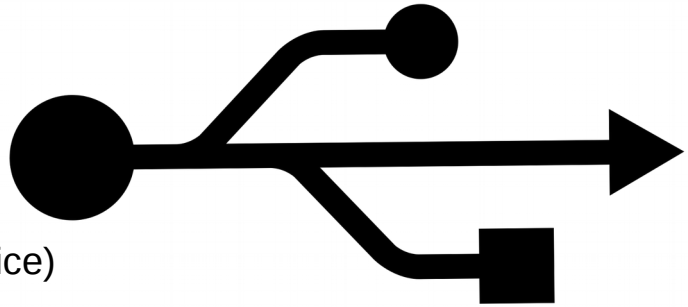
Protocole HDLC

- Protocole de transmission synchrone 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
  - Bourrage d'octet (byte-stuffing) pour la transparence
  - 07Eh est le “fanion” (délimiteur)
  - 07Dh est le caractère d'échappement
  - Le/les fanions sont masqués (transparence) en préfixant par le caractère d'échappement et en inversant le bit 5.

Protocole HDLC



Bus USB (<=2.0)



- Carte d'identité:
  - Bus série asynchrone
  - Topologies supportées:
    - Point → Multi-points (avec un HUB), maître(host) → esclaves (device)
  - Half-duplex
  - 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' : 2.8 → 3.6 en basse vitesse (USB1.0), -10mv → 10mv en haute vitesse
    - 'K' : 0 → 0.3v en basse vitesse (USB1.0), 360mv → 440mv en haute vitesse
    - '0' codé par une transition d'état, '1' codé par aucune transition.
    - Alimentation du périphérique (device) par le bus
      - 5v 500ma jusqu'a USB2.0
      - 5v 900ma à partir de USB3.0

Bus USB

- Permet l'adressage dans le périphérique (adressage logique)
  - Endpoints
- 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

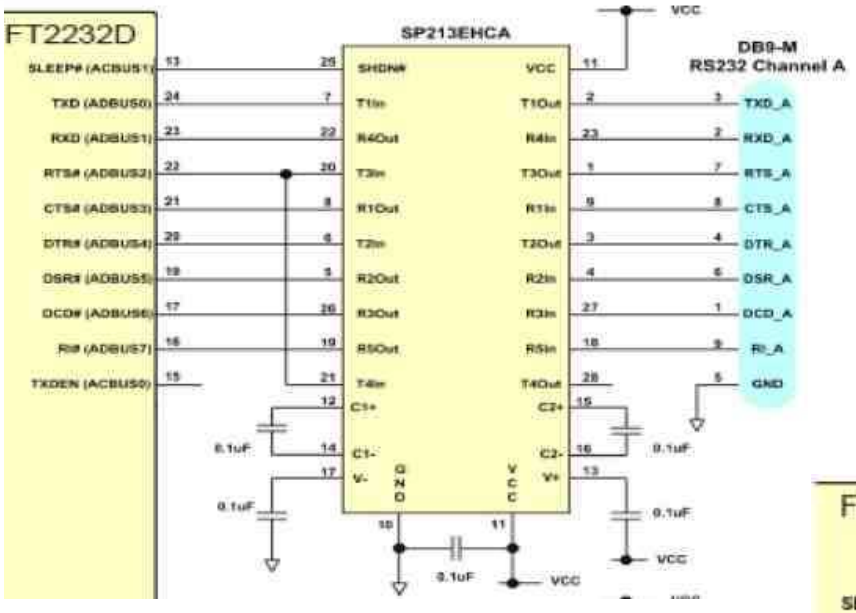
Composants de conversion USB

- FTDI, Silicon-Labs, Cypress
  - Convertit le standard USB vers: UART, I2C, SPI, bit-bang, FIFO matérielle…
  - Pas besoin d'acquérir un USB vendor-id

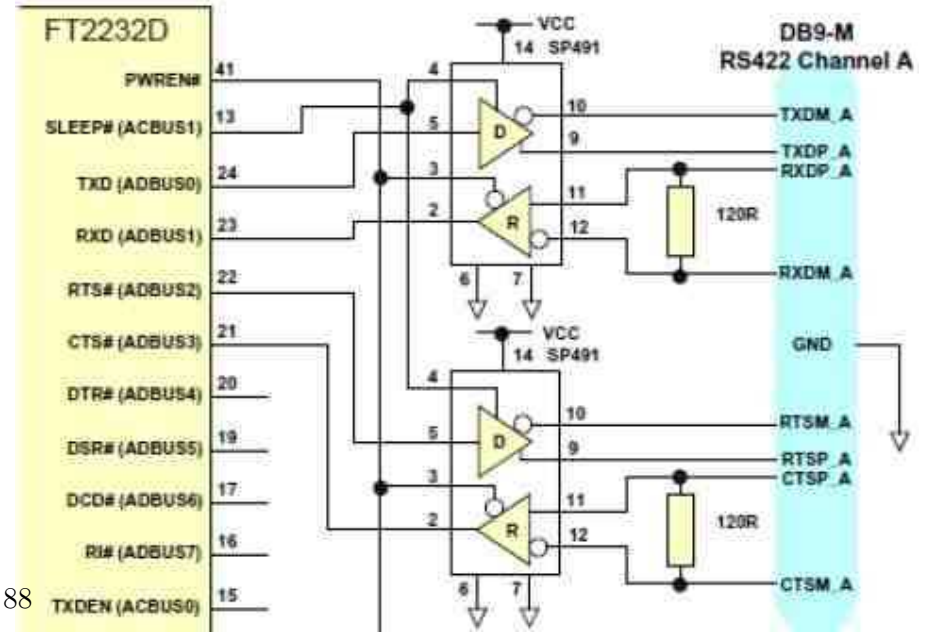
CH340 USB<->TTL

FTDI2232H USB2.0 HS USB<->TTL

Composants de conversion USB



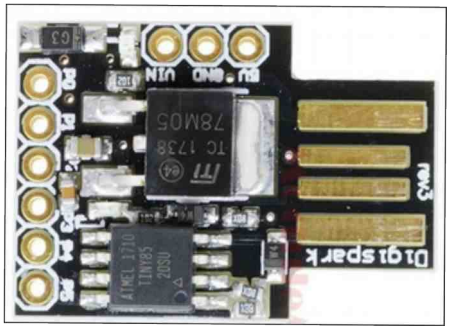
Cablage en convertisseur USB<->RS232



Cablage en convertisseur USB<->RS422

USB sur microcontrôleur

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



Carte microcontrôleur Digispark

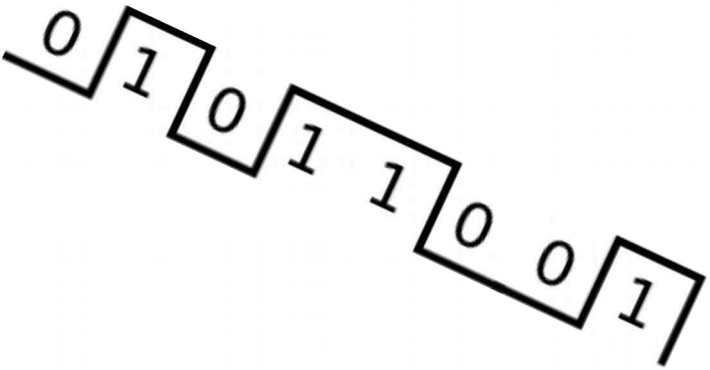
- USB device/host intégré (TI, Atmel, Freescale, PIC ...):
  - Jusqu'à USB2.0
  - Intègre un PHY USB
  - Intègre une stack USB
  - Stack USB logicielle fournie par le fabricant (alternatives open-source)
  - Vendor-ID fournit par le fabricant pour le prototypage

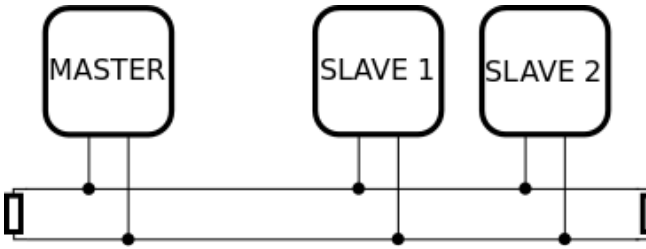
Descripteur USB

- 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



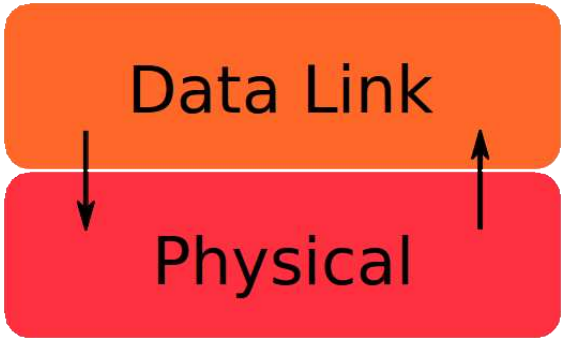


Bus Série Synchrone

- 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

Bus SPI

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



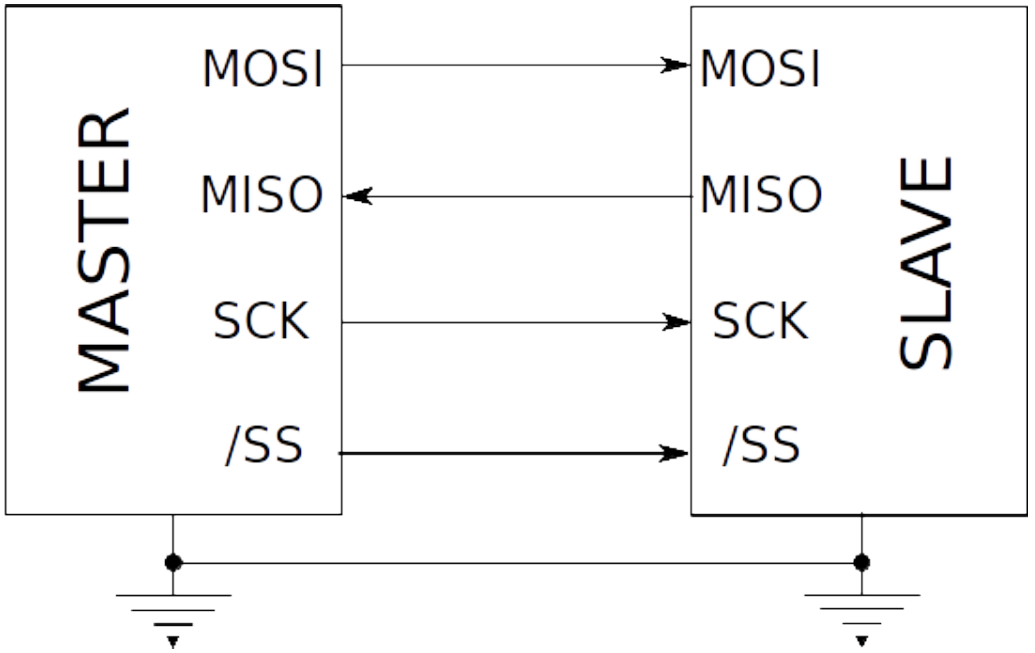
Bus SPI

- 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
- Longeur de lignes:
  - Principalement utilisé comme bus de carte,
  - Longueur centimétrique

Bus SPI

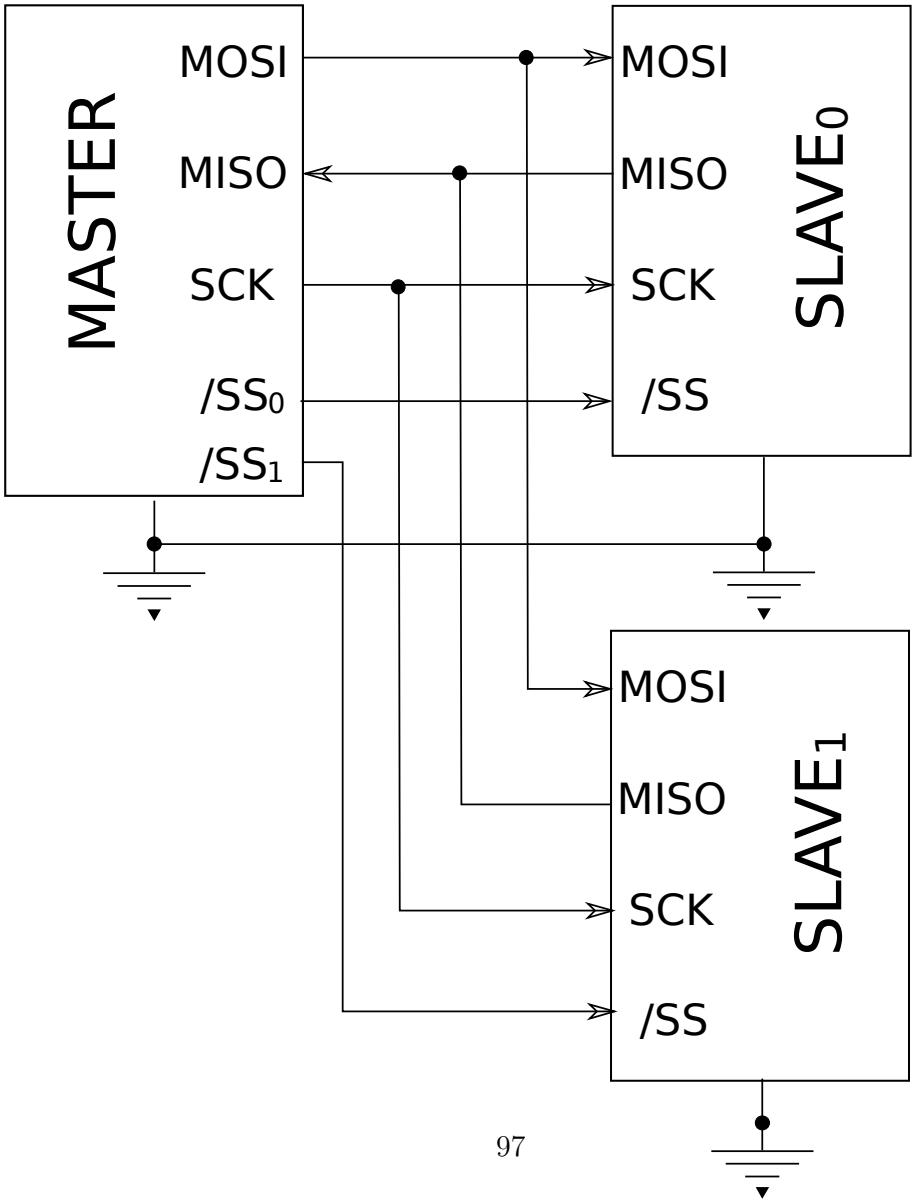
- Implémentation matérielle
  - Registre à décalage en émission et réception
  - Enable du registre sur SS

Bus SPI

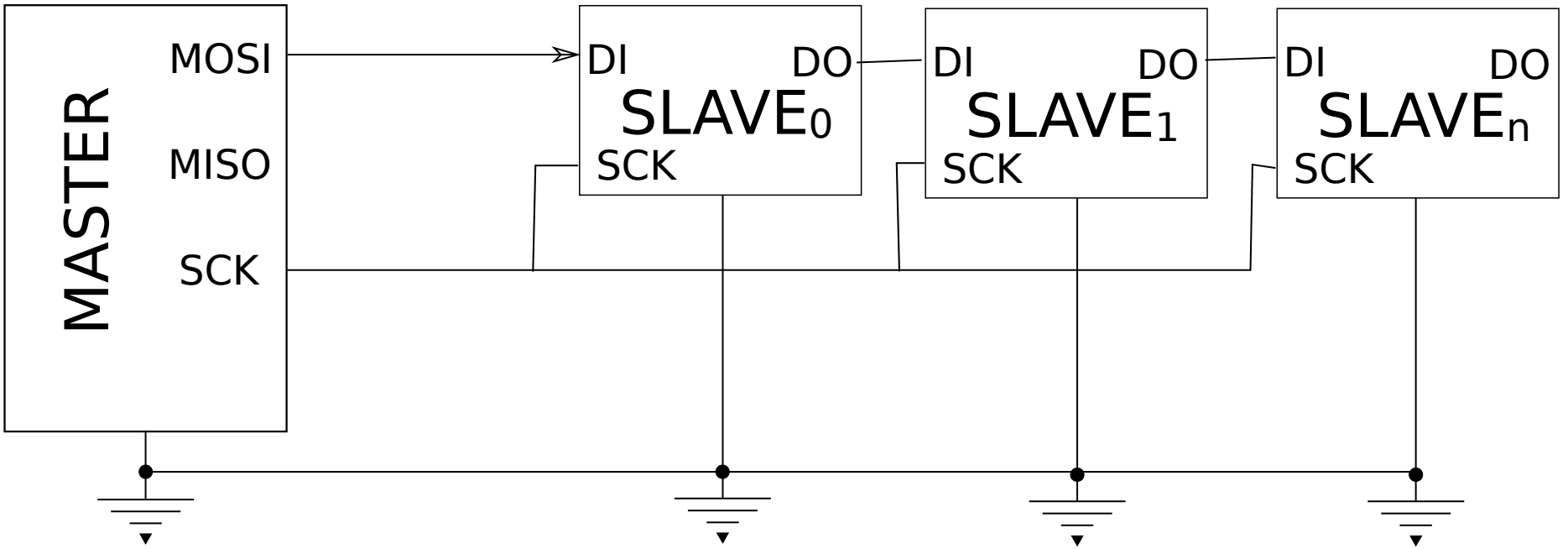


MOSI : Master Out Slave In, synonymes : DO (Data Out) coté maître DI (Data In) côté esclave
MISO : Master In Slave Out, synonymes : DO (Data Out) coté esclave DI (Data In) côté maître
SCK : Serial ClocK, horloge de transmission
/SS : Slave Select, synonymes : /CS (Chip Select)

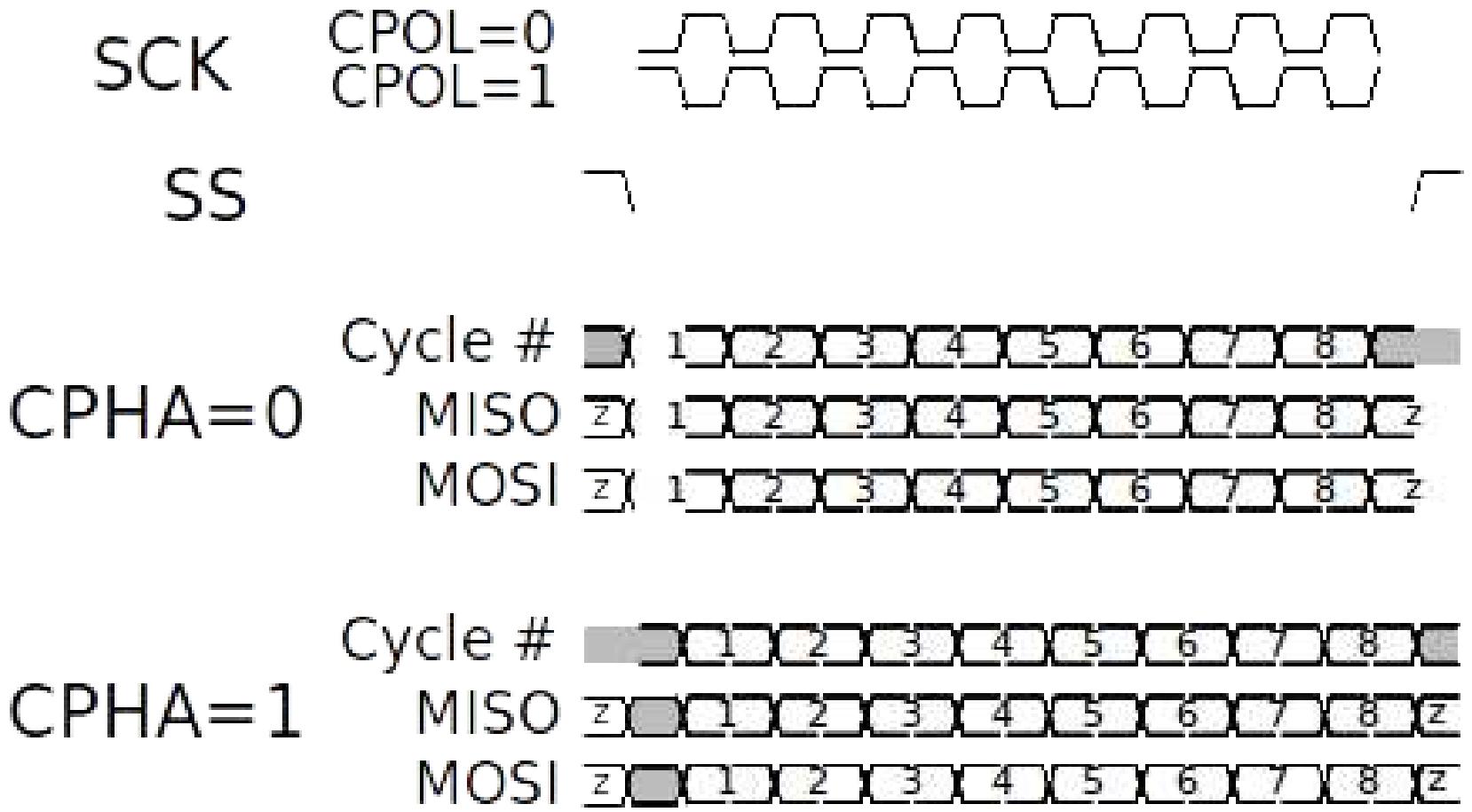
Bus SPI


MASTER
MOSI
MISO
SCK
/SS0
/SS1
SLAVE0
MOSI
MISO
SCK
/SS
SLAVE1
MOSI
MISO
SCK
/SS

Bus SPI


MASTER
MOSI
MISO
SCK
DI
DO
SLAVE0
SCK
DI
DO
SLAVE1
SCK
DI
DO
SLAVEn
SCK

Bus SPI



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

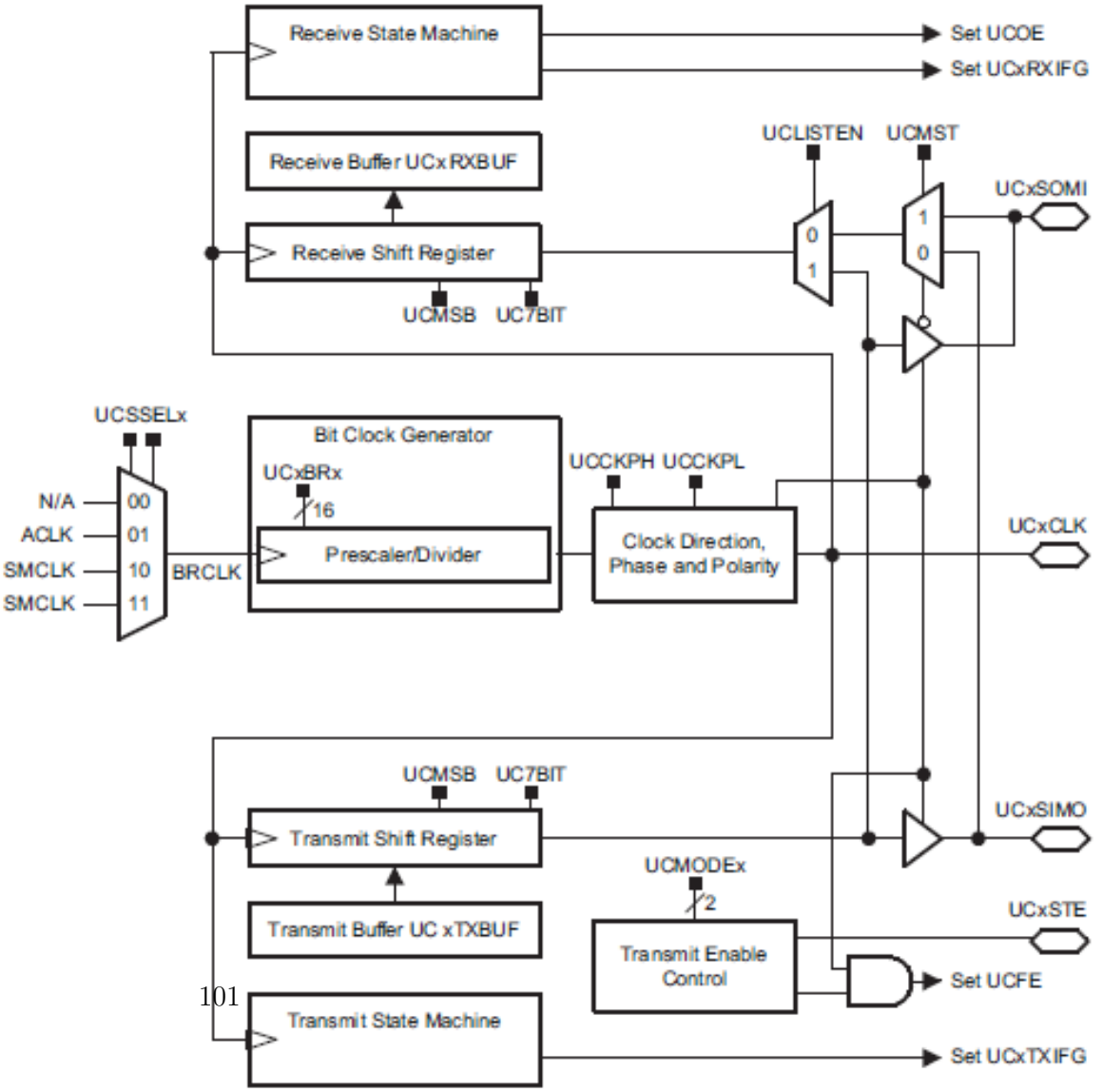
Bus SPI

- Configuration:
  - Fréquence
  - Phase de l'horloge (échantillonnage sur front montant/descendant)
  - Etat de l'horloge au repos (Polarité)
  - Configuration du/des Slave Select

Bus SPI

- Configuration périphérique intégré
- Ex: microcontrôl

  TI 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(spi_slave * 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] = spiWriteByte(send_buffer[i], slave);
	}
	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 ;
        unsigned char inBuf = 0 ;
        __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);
}
```

Bus SPI

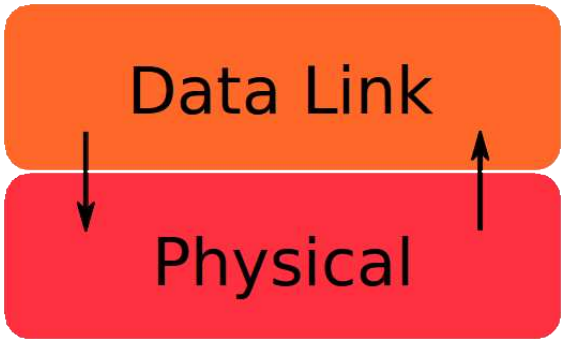
```
Void transfer_data_spi(uchar addr, char * snd, char * rcv, uint length){
        clr_ss(addr);
        delay(N);
        for(i = 0 ; i< length ; I ++){
                rcv[i] = transfer_byte_spi(snd[i]);
        }
        delay(N);
        set_ss(addr);
}
```

Bus SPI

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

Bus I2C

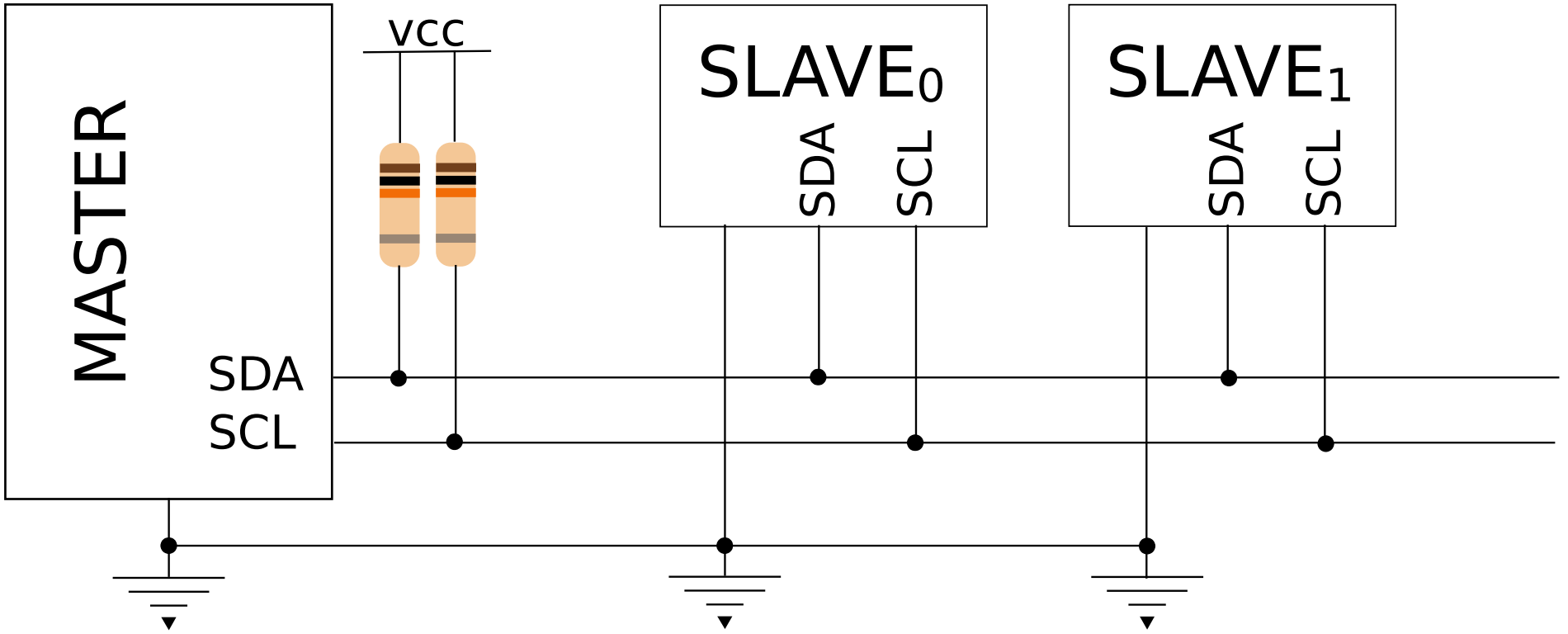
- I2C (prononcer “I square see”)
- Carte d'identité:
  - Bus série synchrone
  - Topologies supportées:
    - Point → multi-points en maître/esclaves, multi-maîtres/esclaves
  - Half-duplex



Bus I2C

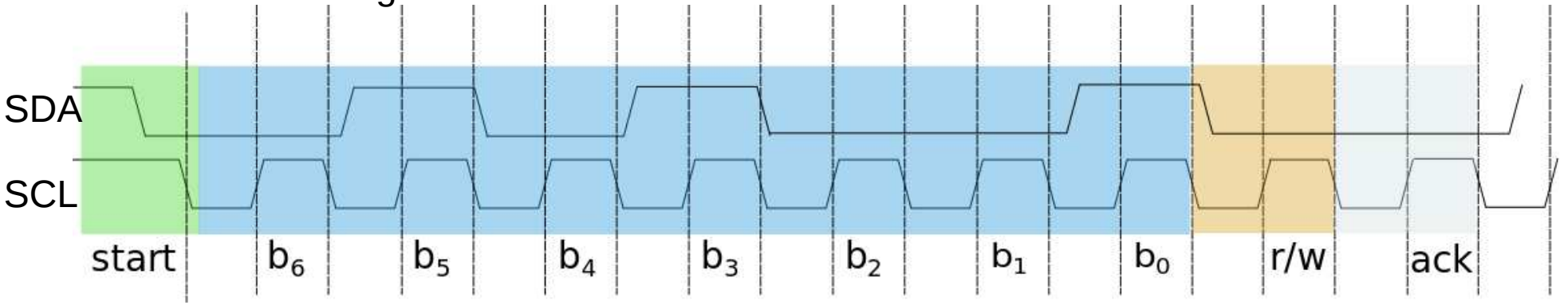
- Spécification électriques:
  - Bus sur collecteur ouvert : deux niveau logiques : 0, Hiz (haute impédance)
- Câblage:
  - 3 conducteurs : SCL, SDA, GND
- Vitesse:
  - 100KHz, 400 kHz (fast-mode), 1MHz (fast-mode plus), 5MHz (ultra fast-mode)
- Longueur de lignes:
  - Bus de carte et bus de terrain
  - 100m max, dépend de l'impédances du câblage

Bus I2C



Bus I2C

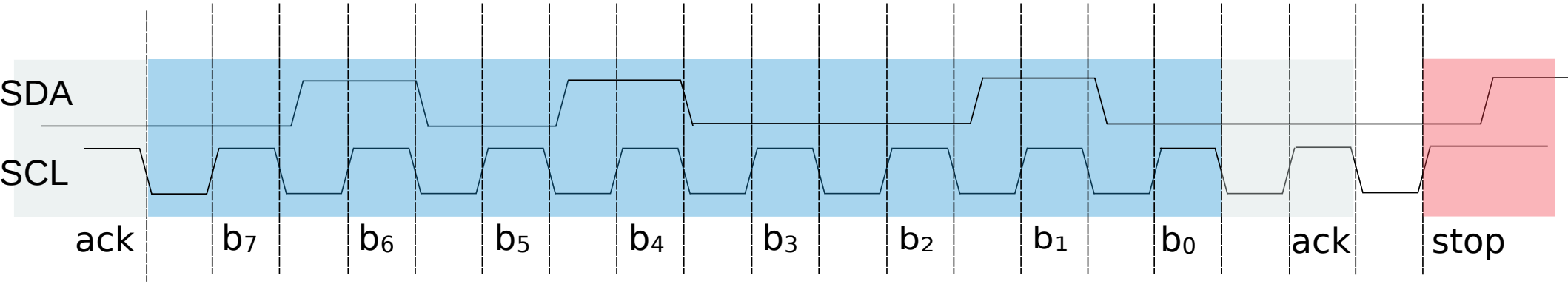
Phase d'adressage :



1) Envoi de la condition de start par le maître
2) Envoi des 7 bits d'adresse
3) Envoi du bit de lecture (bit à 1) ou d'écriture (bit à 0)
4) Réception de l'acquitement: 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 ou un esclave correspondrait à l'adresse envoyée, celui-ci forcera le bus au niveau bas.

Bus I2C

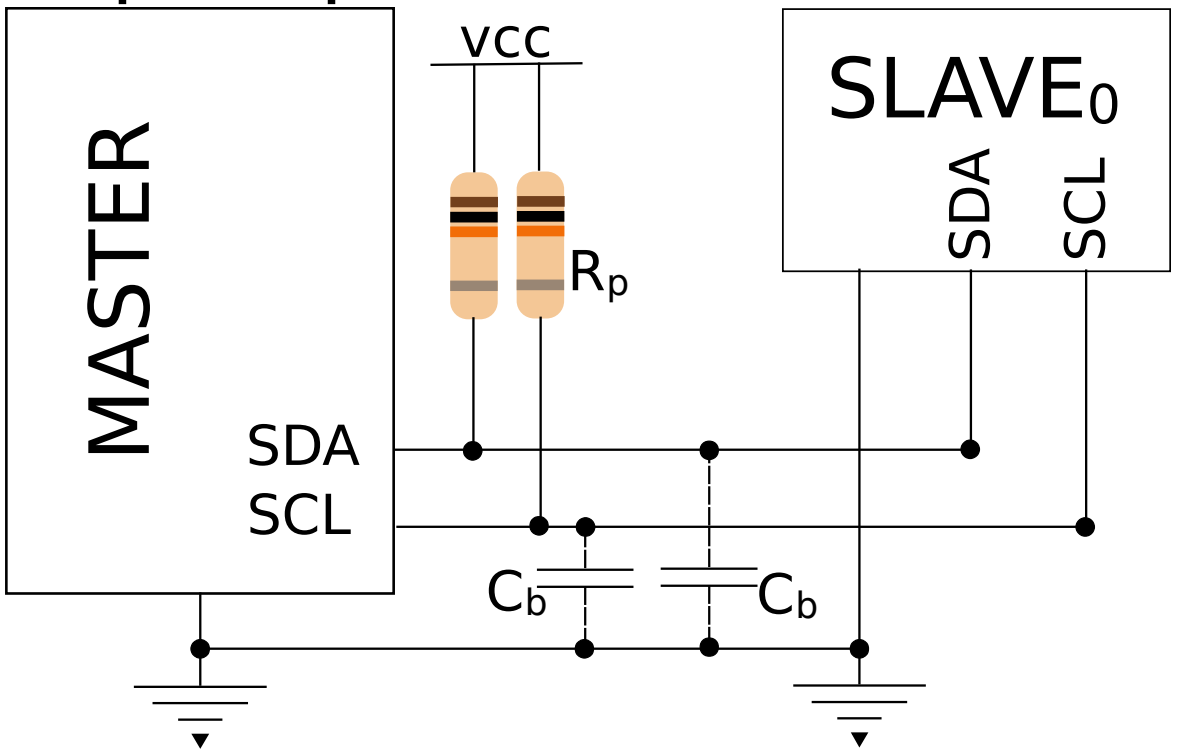
Phase d'envoi de données :



1) Réception de l'acquitement de la donnée précedente ou de la phase l'adressage.
2) Envoi des 8 bit de données
3) Réception de l'acquitement
4)Si fin de la communication envoi de la condition de stop. Si la communication continue, la trame se répète.

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

Bus I2C

- Acquitement
  - Acquitement de l'adresse par l'esclave
  - Acquitement de la donnée écrite par l'esclave
  - Acquitement de la donnée lue par le maître
  - Pas d'acquitement ...

Bus I2C

- Acquitement
  - Acquitement de l'adresse par l'esclave
  - Acquitement de la donnée écrite par l'esclave
  - Acquitement de la donnée lue par le maître
  - Pas d'acquitement ...

Bus I2C

- Rôle Acquitement
  - Acquitement de l'adresse : périphérique existe (scan du bus)
  - Acquitement de la donnée écrite : Donnée bien interprétée.
  - Acquitement de la donnée lue : Donnée bien lue, la communication continue.
  - Mécanisme d'auto-incrémentation d'adresse logique (voir page suivante): Lors de l'acquitement pour une lecture ou une écriture vers une mémoire, l'adresse est automatiquement incrémentée pour pouvoir enchainer une lecture ou une écriture.

I2C

- Adressage logique
  - Adressage d'un registre dans le composant
  - Ecriture:
    - Envoie de l'adresse du registre, puis de/des données
  - Lecture:
    - Solution 1 : Un accés en écriture puis un accés en lecture
    - Solution 2 : Renvoi de la condition de start sans condition de stop (SMBUS)

I2C

- Périphérique intégré
- TI MSP430

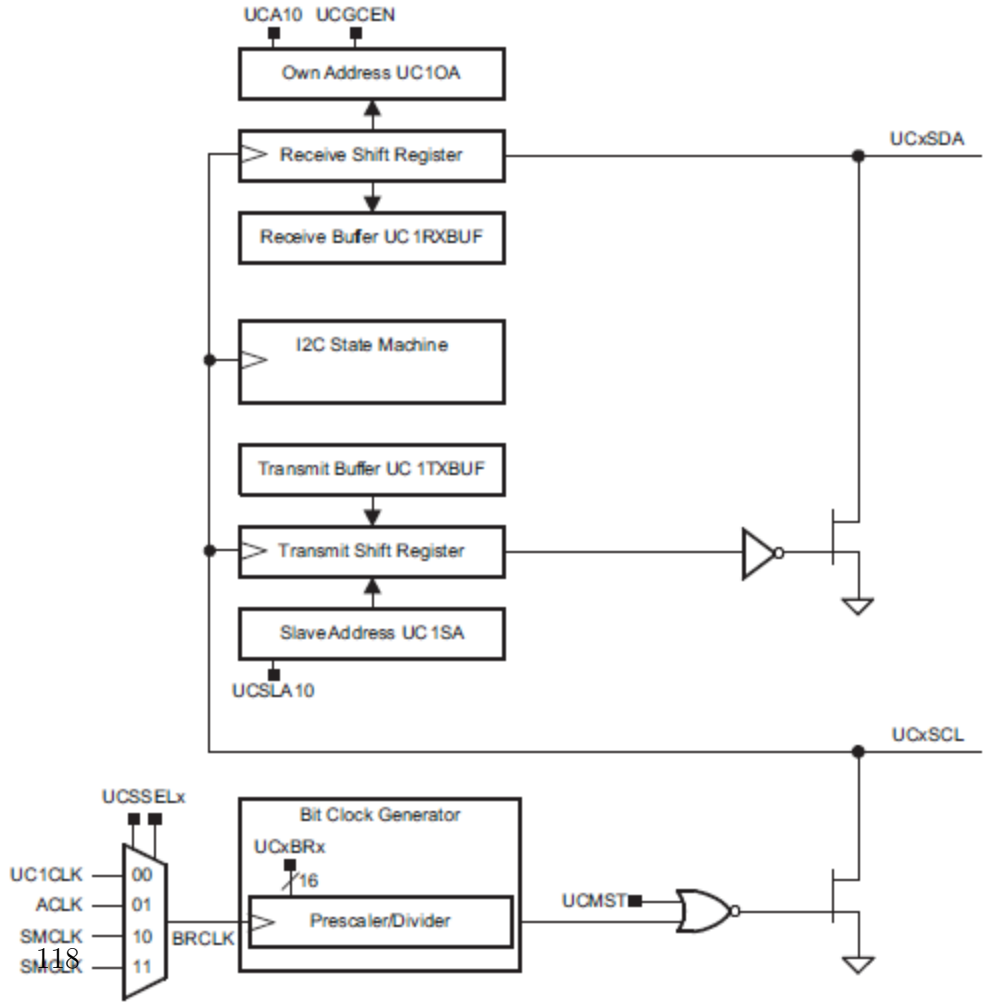


Figure 17-1. USCI Block Diagram: I²C Mode

I2C

```
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

```
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
  - Besoin d'un buffer collecteur ouvert pour SDA
  - Certains micro-controlleur, sortie en collecteur ouvert quand la GPIO est configurée en entrée

I2C

```
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);
        }
        deturn d ;
}
```

I2C

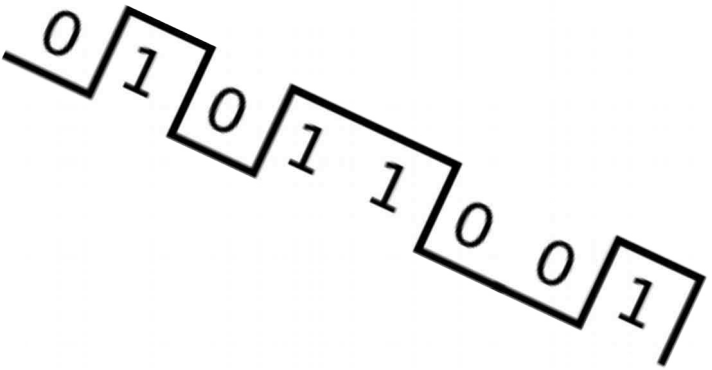
```
bit write_data(uchar addr, uchar * data, uint length){
        Start();
        write_byte(addr << 1);
        If(!read_ack()){
                stop();
                return 0 ;
        }
        for(i = 0 ; I < length ; I ++){
                write_byte(data[i]);
                If(!read_ack()){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

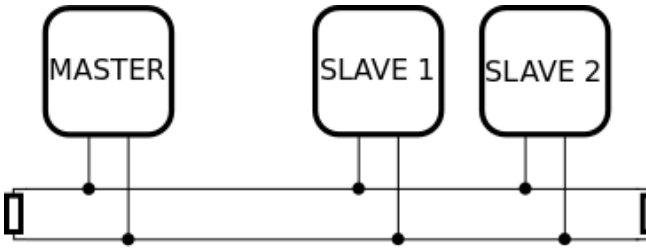
```
bit read_data(uchar addr, uchar * data, uint length){
        start();
        write_byte((addr << 1) | 0x01);
        If(!read_ack()){
                stop();
                return 0 ;
        }
        for(i = 0 ; I < length ; I ++){
                data[i] = read_byte();
                write_ack();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

I2C

- Exemples de périphériques
  - EEPROM
  - HDMI (i2c pour configurer l'écran)
  - Sonde de température
  - Capteurs inertiels

Bus parallèle asynchrone:
Bus Centronics
Bus VME
Bus PC104





Bus parallèle

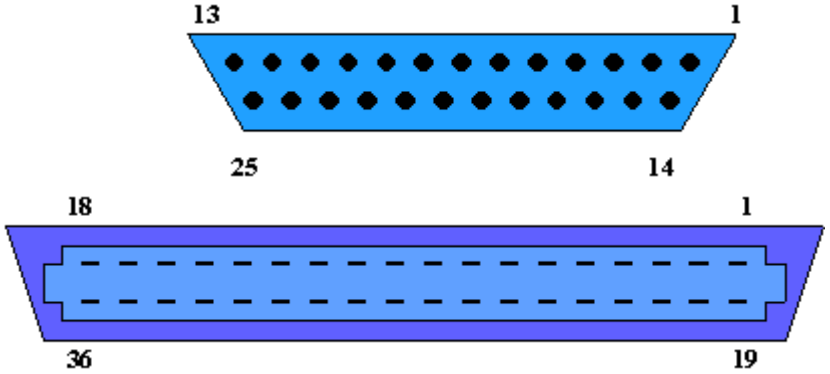
- Le bus transmet les bits d'un mot en parallèle
- Les hôtes communiquent quand ils le veulent/peuvent.
- Avantage:
  - Vitesse du bus
- Inconvénients:
  - Câblage (nombre de conducteurs et distance)
  - Sensible au bruit

Bus centronics

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

Bus centronics

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



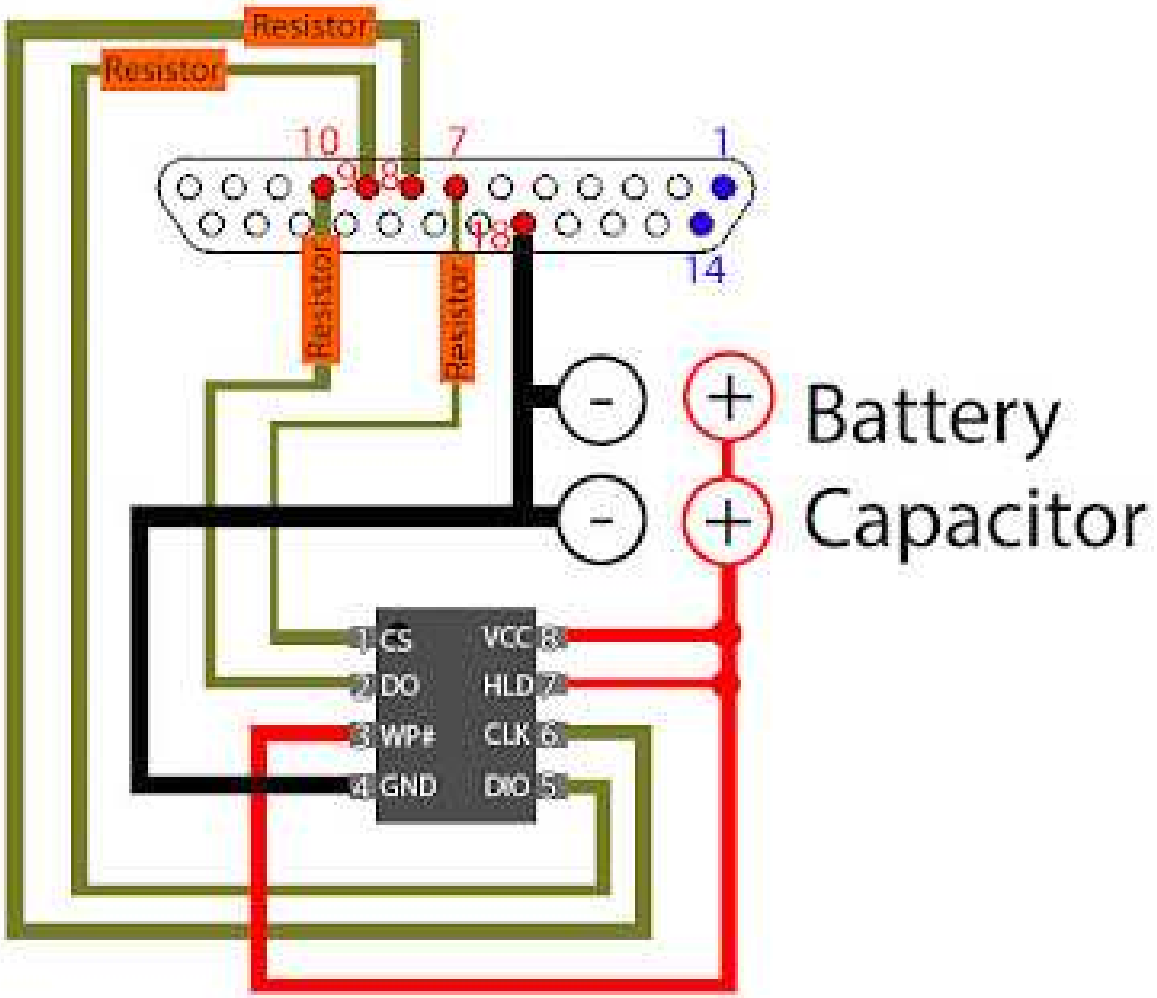
Bus centronics

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 centronics

- Bit du port tous adressables
- Contrôle total en logiciel
- Permet d'émuler des bus de communication “lents”
  - SPI, I2C
  - LCD
  - Protocole de programmation

Bus centronics



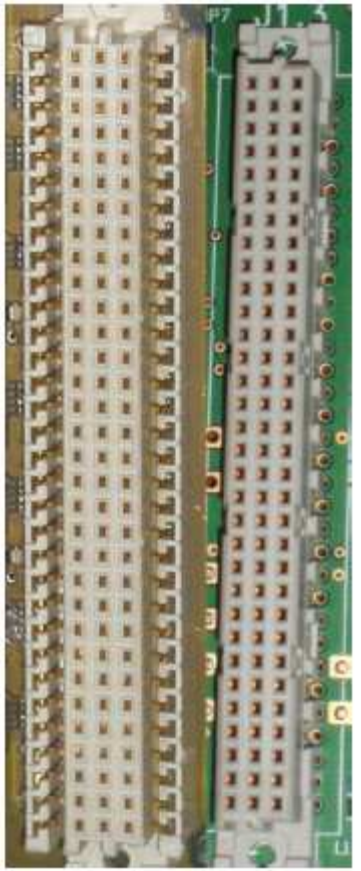
Bus VME

- 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 VME

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

Bus VME



Bus VME

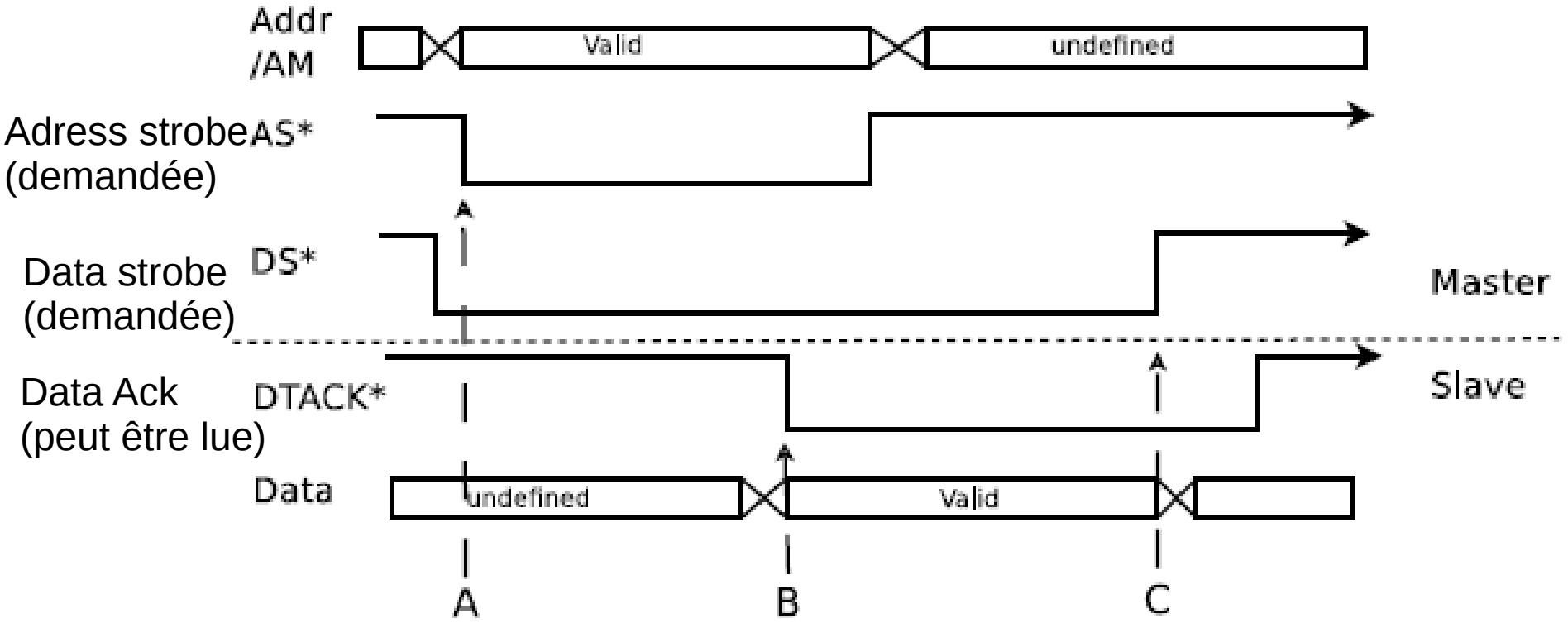

3U
6U
9U
Size A
Size B
Size C
Size D
100mm
160mm
233mm
160mm
233mm
340mm
366mm
340mm
U.S. Technologies - vmebusdirect.com

Bus VME

- Bus d'addresse et bus de données
  - Addresses : 16/24/32 bits
  - Données: 8/16/32
  - 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 VME

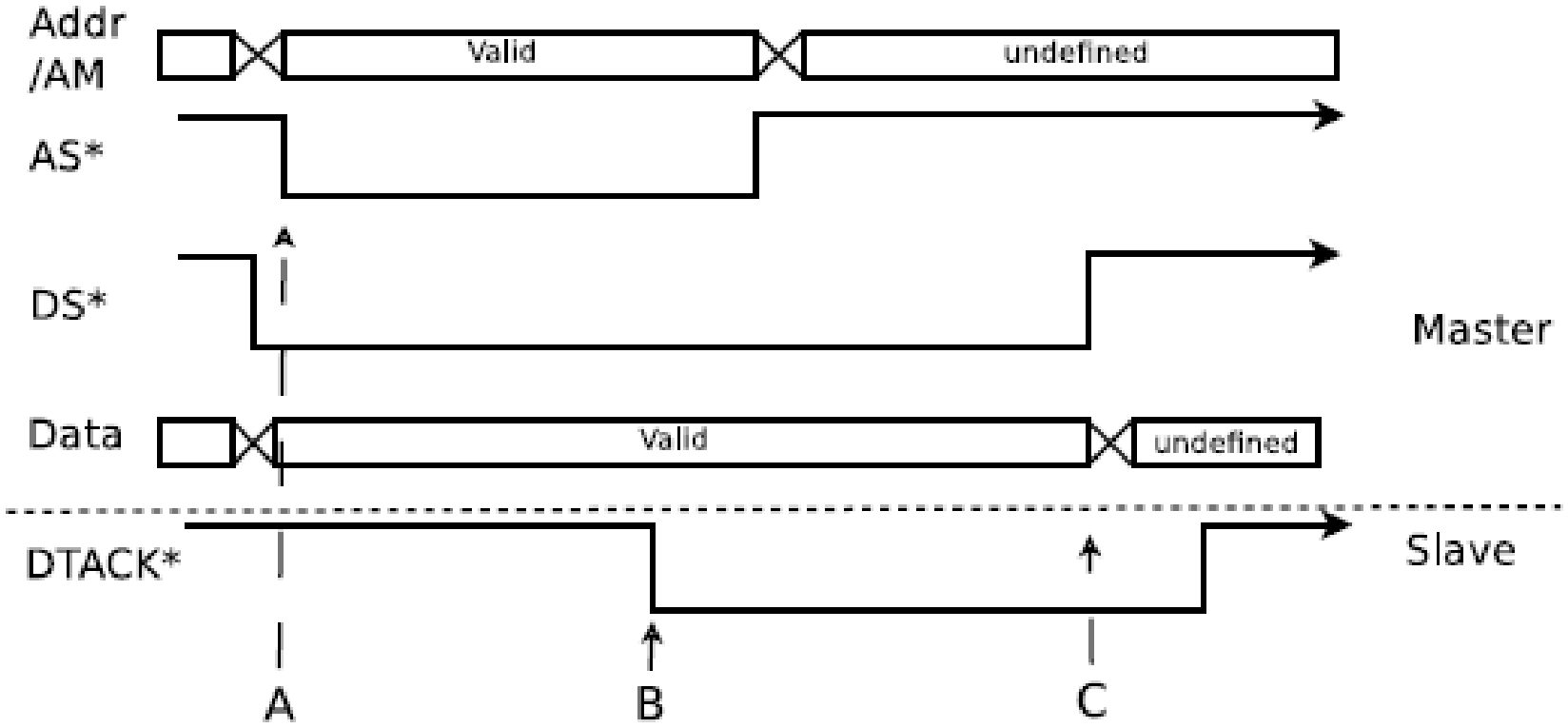


A Master requests data

B Slave provides data

C Master ends cycle

Bus VME



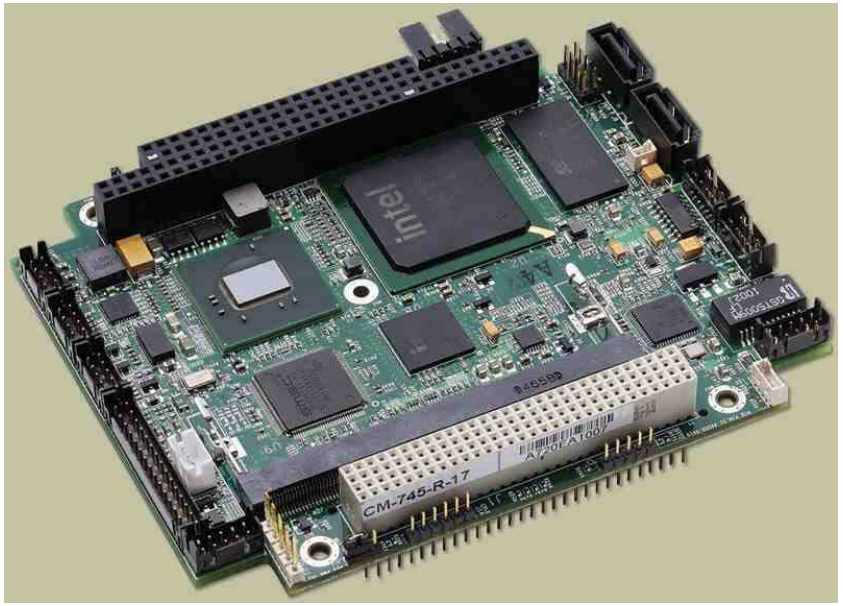
A Master provides data

B Slave accepts data

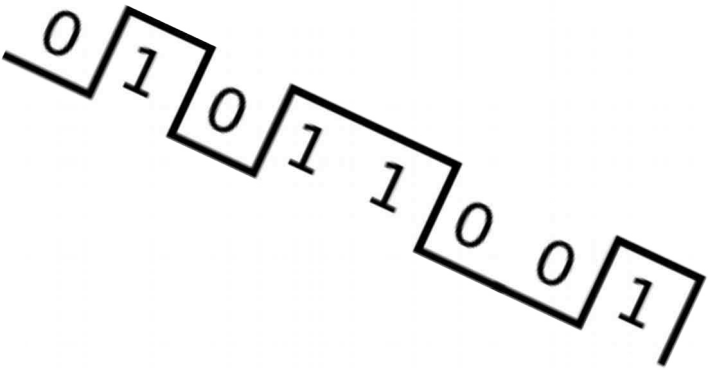
C Master ends cycle

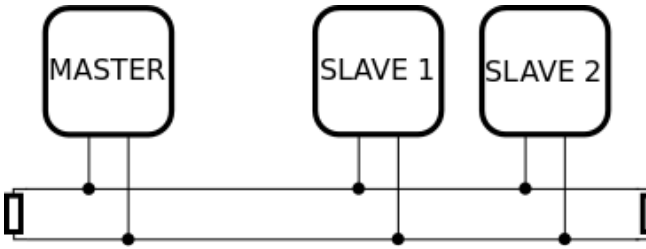
BUS PC104

- 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)*



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





Système GPS

- Mire de satellites en orbite à 22 000km
- 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)
  - Envoit la position à l'application sur liaison série (1 à 10Hz)
  - Format de communication le plus répandu : NMEA
  - Ajouter Schéma de principe (MCU

Système GPS

- Trame NMEA
- Séquence de caractères ASCII
- Commence par $ suivi du nom de trame
  - GPMRC, GPGGA, GPRMA …
- GPRMC = Infos utiles de localisation

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

Système GPS

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

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 (voir note1).

4 = (N) Nord ou (S) Sud

5 = Longitude en degrés/minutes (voir note1).

6 = (E) Est ou (W) Ouest

7 = Vitesse au sol en nœuds

8 = Cap en degrés

9 = Date

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)

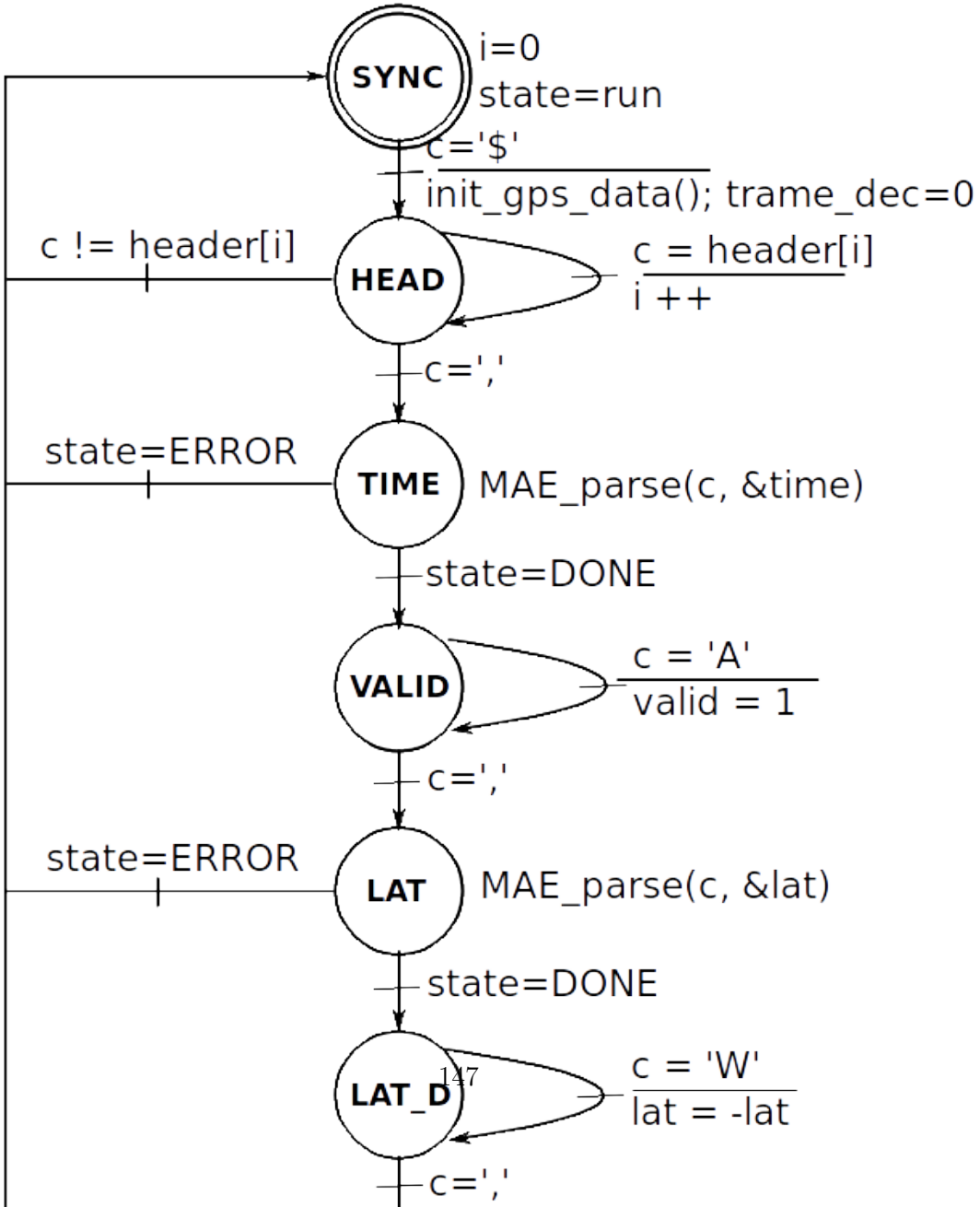
Système GPS

- 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)
  - Stream parsing
    - Traitement des caractères au fur et à mesure de leur réception (peu de mémoire, pas de latence, code plus complexe)

Système GPS

- Stream parsing
  - Modélisation/implémentation par machine à état
  - L'état symbolise le champ en cours
  - Procédure non bloquante, exécutée pour chaque caractère reçu

Système GPS


SYNC
i=0
state=run
c='$'
init_gps_data(); trame_dec=0
c != header[i]
HEAD
c = header[i]
i ++
c=','
state=ERROR
TIME
MAE_parse(c, &time)
state=DONE
VALID
c = 'A'
valid = 1
c=','
state=ERROR
LAT
MAE_parse(c, &lat)
state=DONE
LAT_D
c = 'W'
lat = -lat
c=','

Commandes AT

- Utilisés pour la communication avec un modem (Modulateur/DEModulateur : interface entre deux domaines de communication ex : TTL → HF/RF)
  - Modem bluetooth, GSM, GPRS, ZigBee
- 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)

Commandes AT

- 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)

Commandes AT

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.

Commandes AT

- 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

Commandes AT

- 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»)

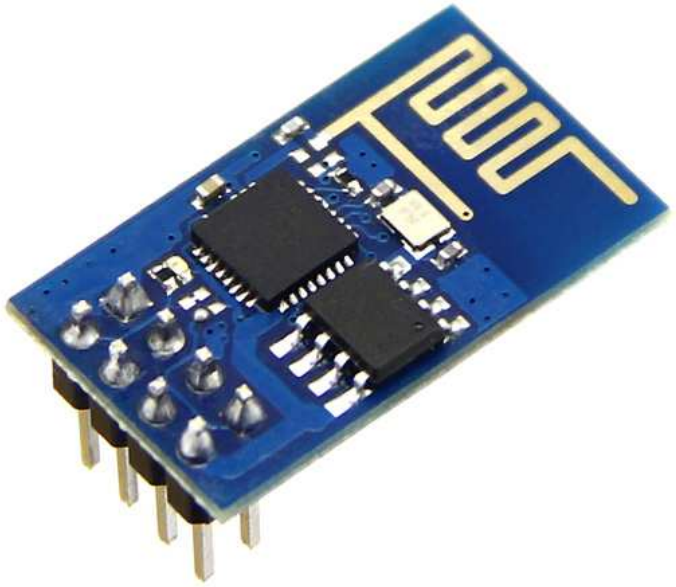
Commandes AT

- Codage 7-bit

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	

Commandes AT

- Modules Wifi
  - ESP8266 : ~5$
  - Wifi 802.11b/gn
  - AP/SLA

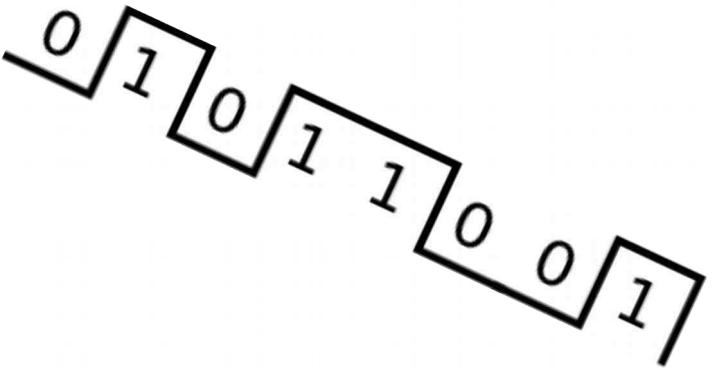


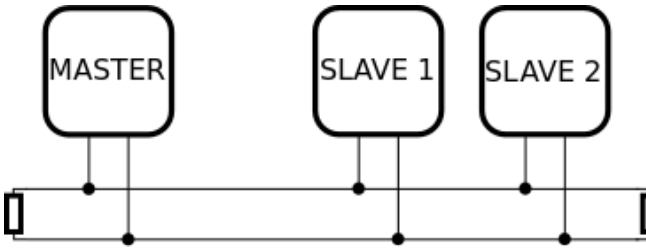
Commandes AT

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



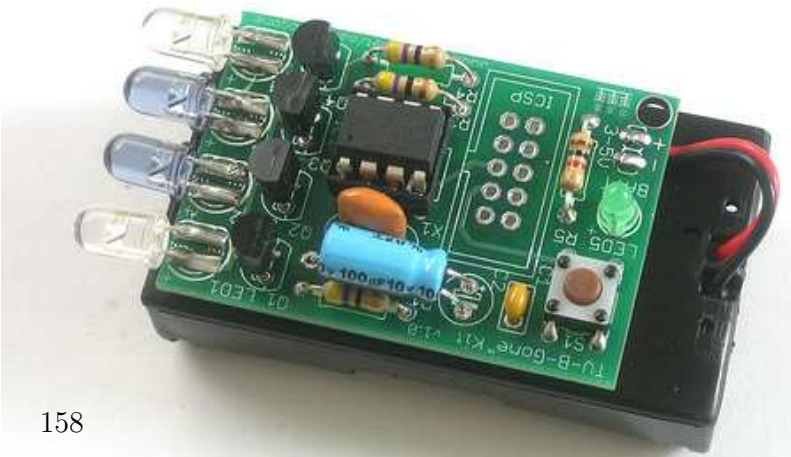


Communication optique

- Le média de communication est le spectre lumineux visible/infrarouge
  - Transmission « dans l'air »
  - 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)

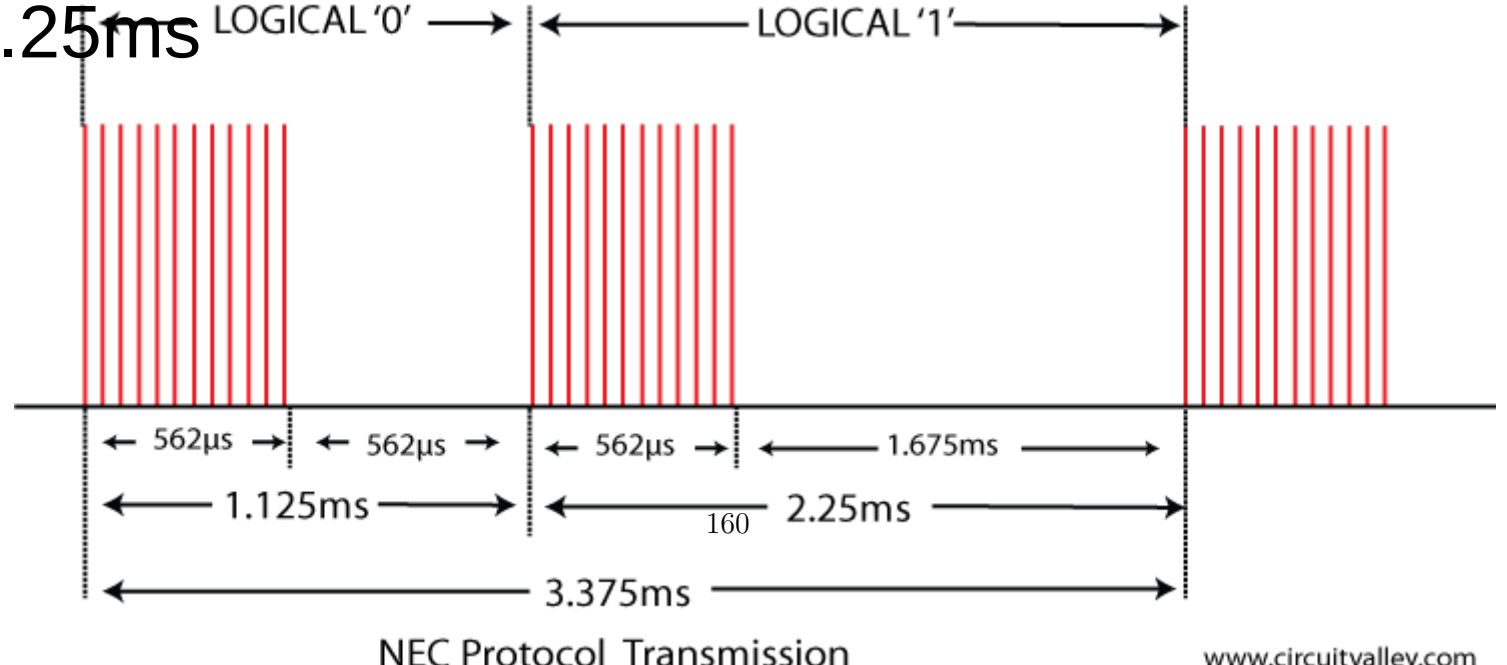


Communication IR

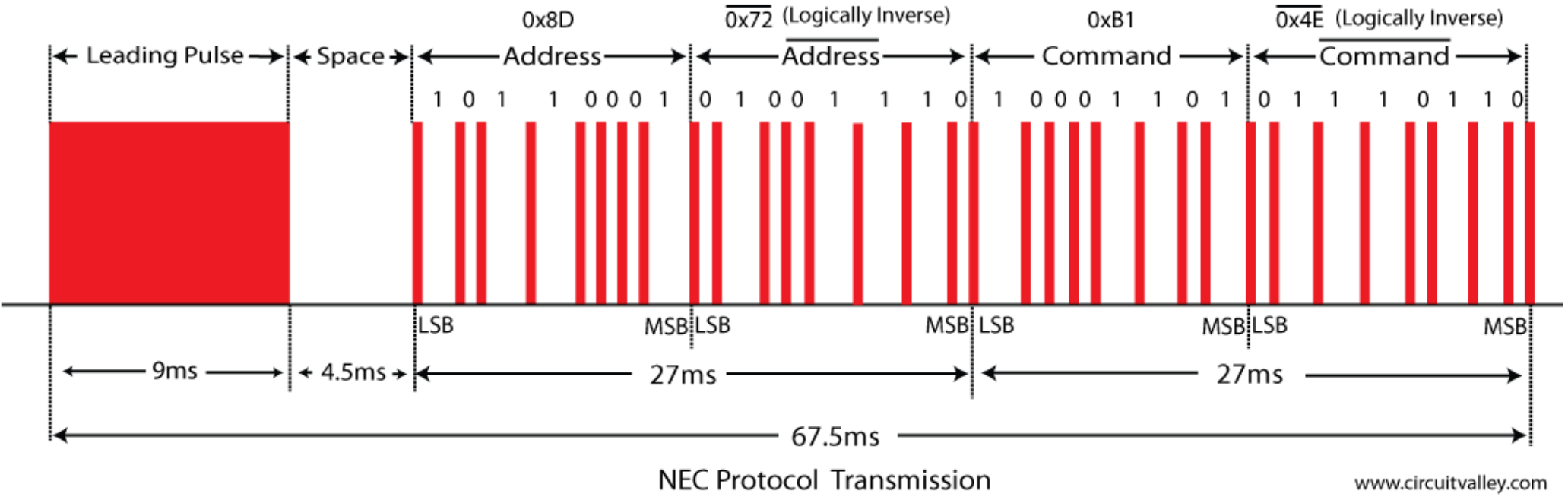
- Protocole NEC
  - A 9ms leading pulse burst (16 times the pulse burst length used for a logical data bit)
  - A 4.5ms space
  - The 8-bit address for the receiving device
  - The 8-bit logical inverse of the address (8 more adressing bits in extended mode)
  - The 8-bit command
  - The 8-bit logical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

Communication IR

- 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



Exemple de trame

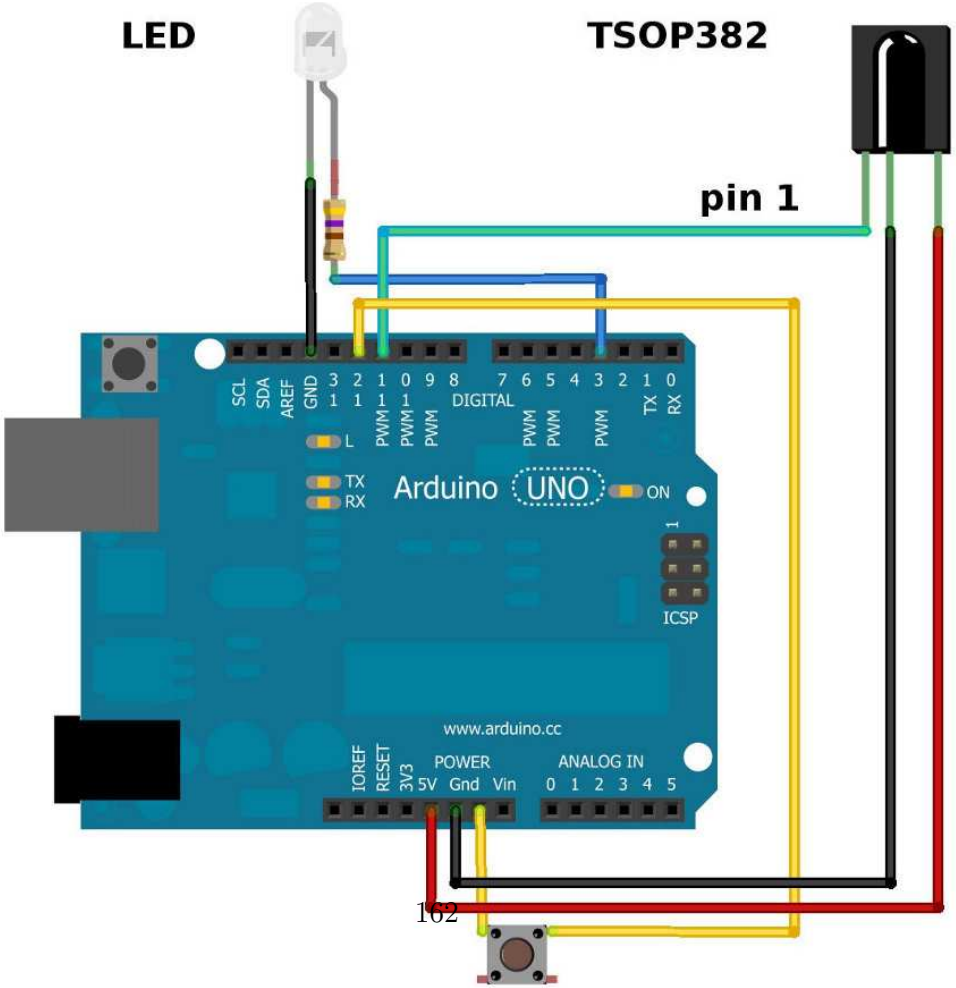


NEC Protocol Transmission

www.circuitvalley.com

Communication IR

- Emetteur récepteur sur Arduino



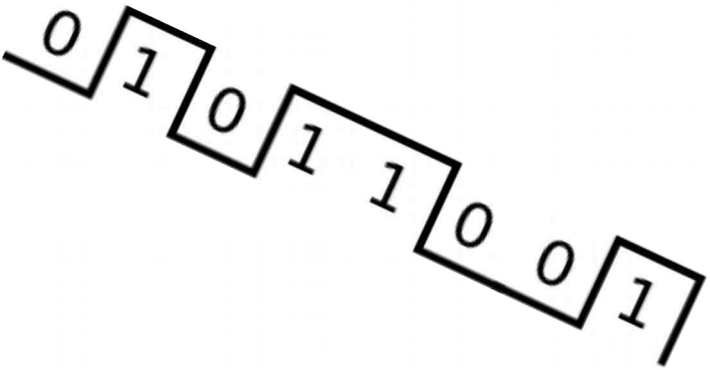
Communication IR

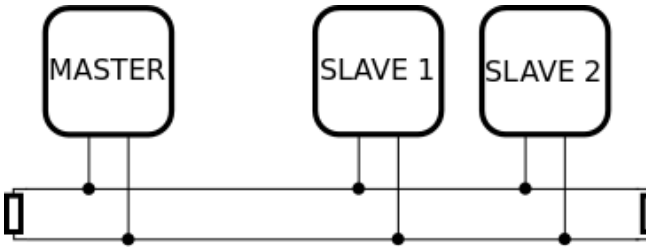
- Les codes des commandes dépendent du constructeur et du modèle.

Li-Fi (Light Fidelity)

- Communication par éclairage ambiant utilisant le spectre visible
- Modulation de l'éclairage ambiant à un fréquence non perceptible à l’œil nu
- Encore en développement

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





Communication MLI

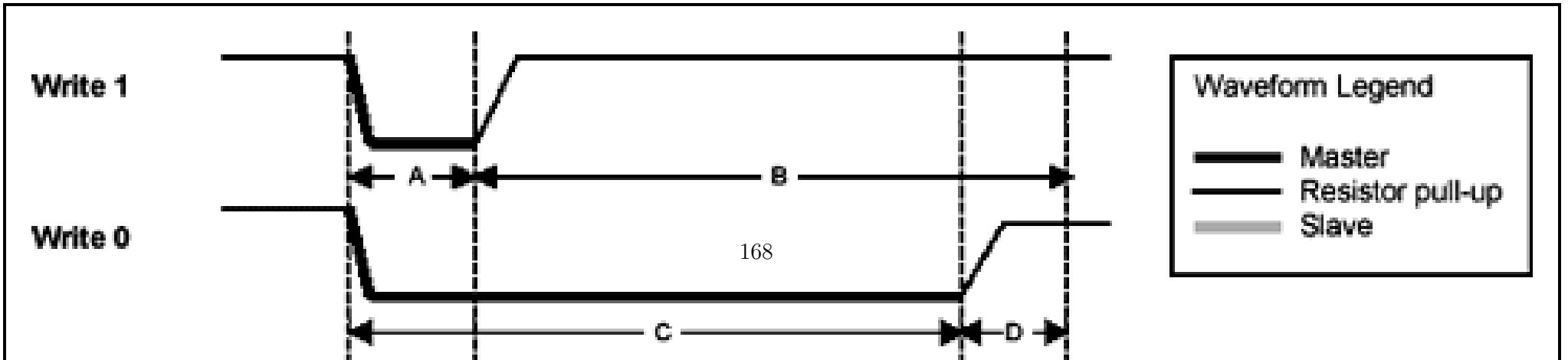
- Modulation par Largeur d'Impulsion : L'information est codée par une durée de maintien à un état ou par le rapport cyclique du signal (plutôt que par niveau)
- Avantages :
  - Pas de synchronisation à récupérer
  - Facilité de mise en œuvre
- Inconvénient :
  - Débit réduit

Protocole OneWire

- Développé par Dallas Semiconducteur
- Vitesse de communication 16.3 kbit/s
- Tension de 3v ou 5v
- Possibilité de 'parasite power ' : alimentation du périphérique par les données

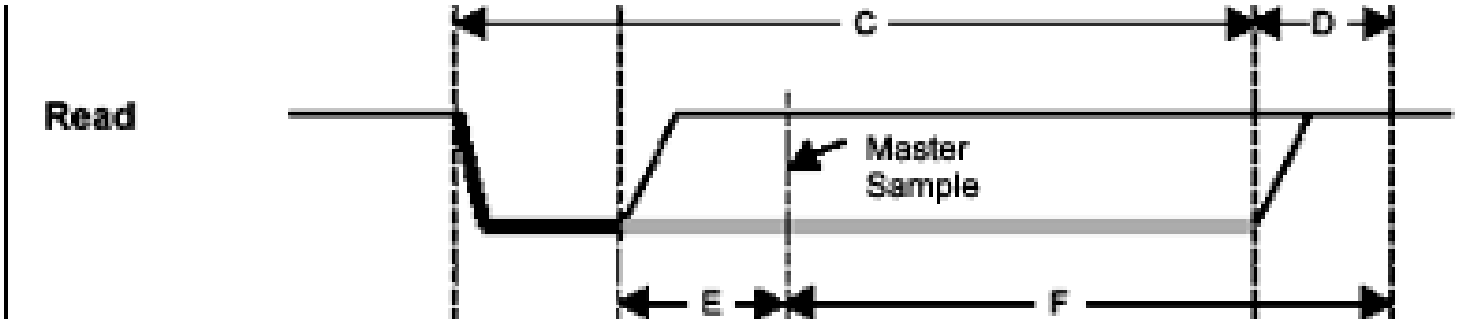
Protocole OneWire

- Emission (période d'un bit 60µs):
  - Emission d'un '1'
    - Mise à l'état dominant de la ligne (0v) pendant 15µs
    - Mise à létat récessif pendant (1v) pendant 45µ
  - Emission d'un '0'
    - Mise à létat dominant pendant 60µs
    - Mise à l'état récessif pendant 1µs à 15µs



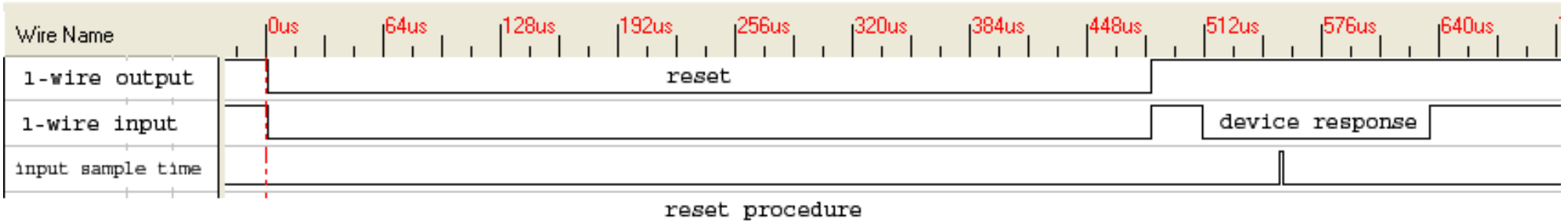
Protocole OneWire

- Réception
  - Emission d'un '1' et lecture de létat de la ligne au bout de 30µs

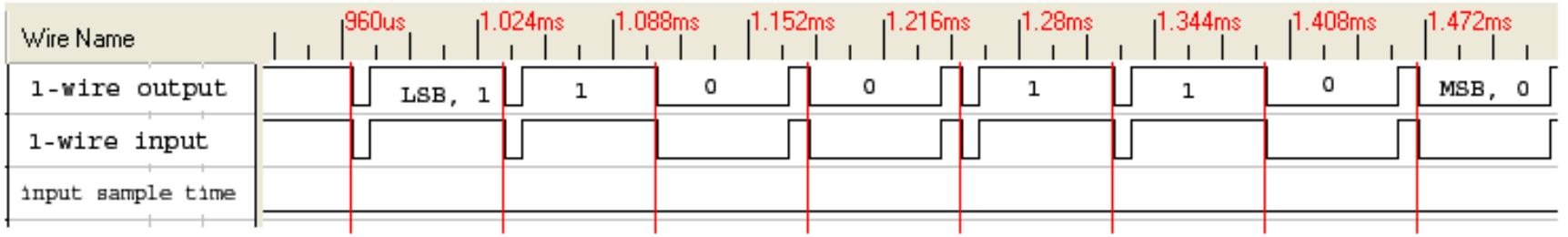


Protocole OneWire

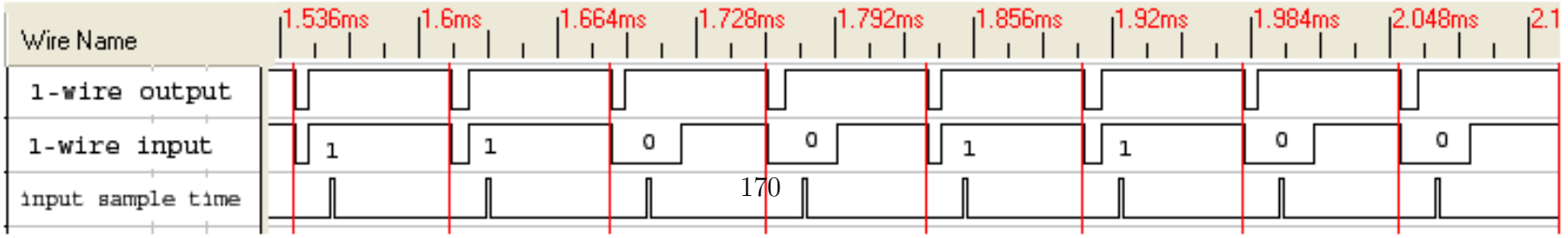
1 Wire reset, write and read example with DS2432



reset procedure



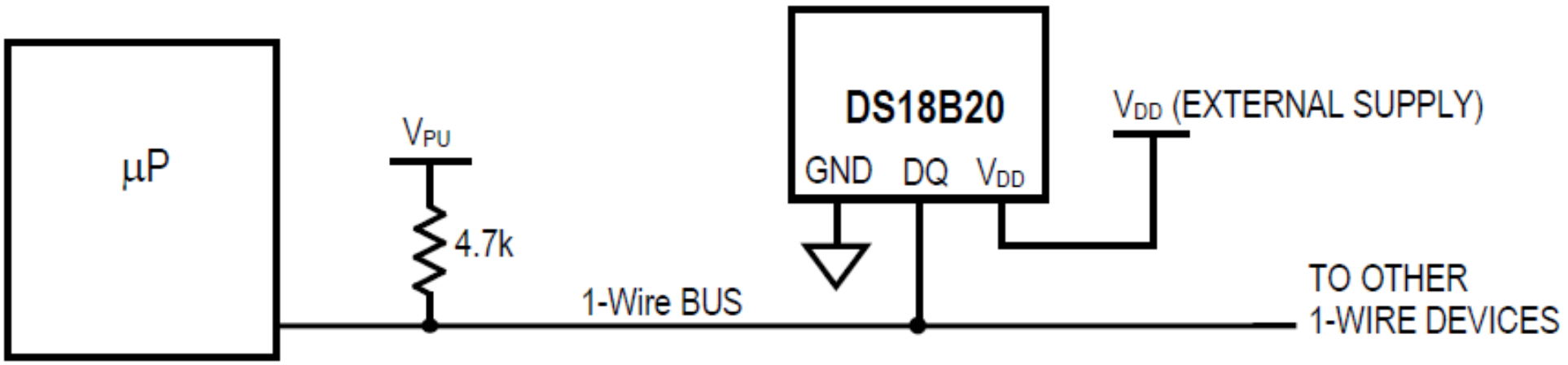
send byte x"33" (b"00110011")



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

Protocole OneWire

- La sonde de température DS18B20



Protocole OneWire

- Lire la sonde DS18B20

Protocole NeoPixel

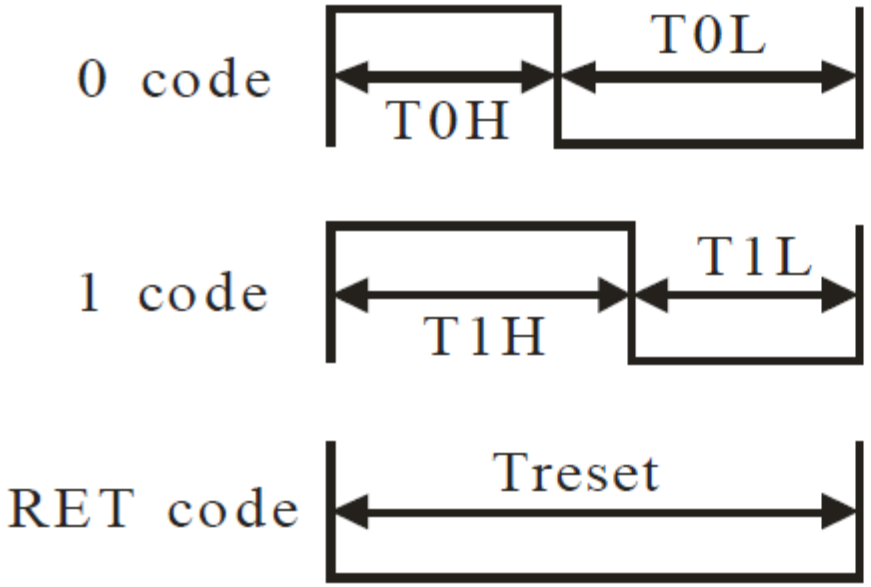
- Protocole de pilotage de LED RGB à contrôleur intégré
- Communication simplex
- Période d'un bit : ~1.30µs
- Débit max : ~800kbps
- Permet le « daisy chaining »

Protocole NeoPixel

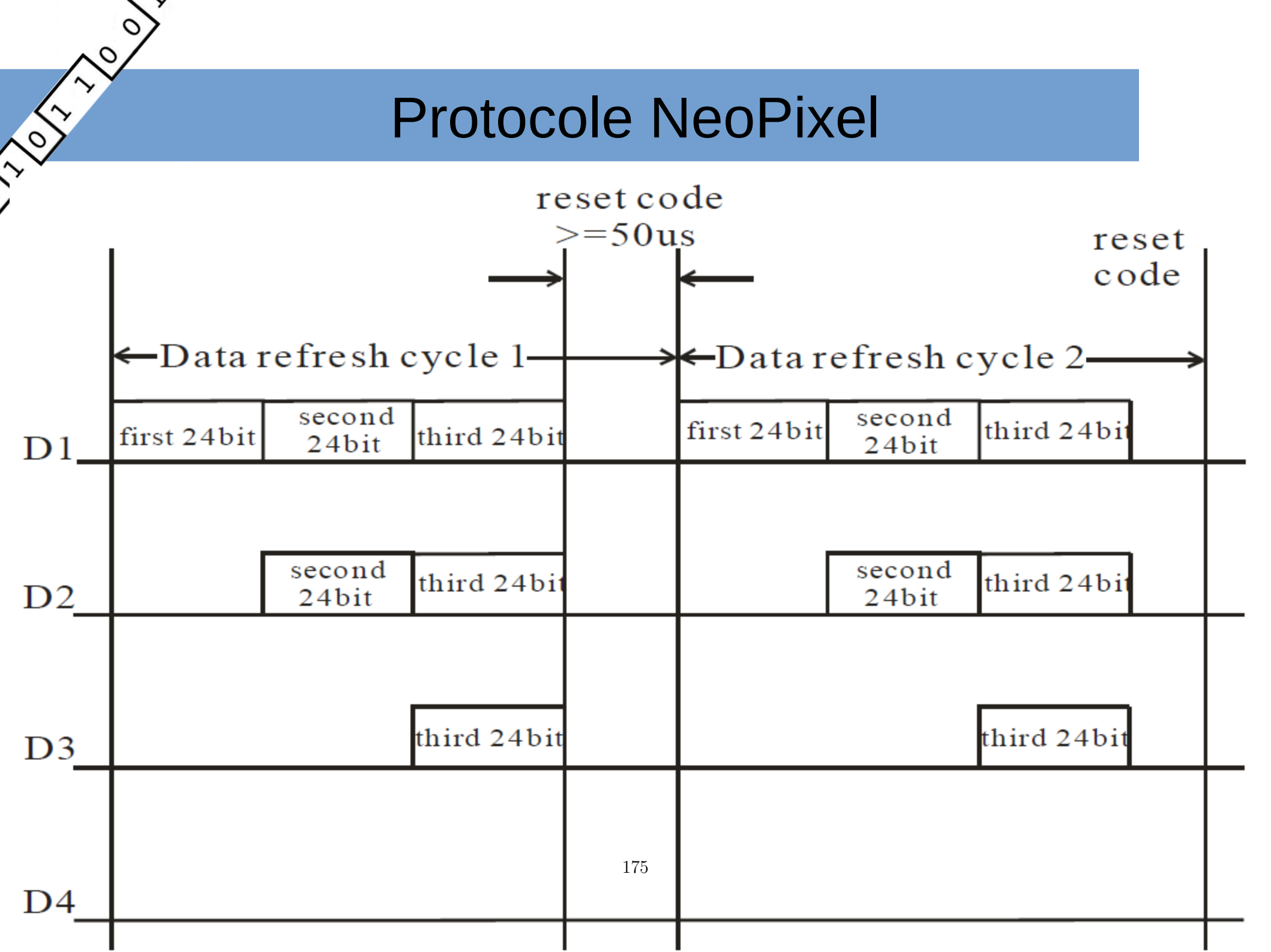
Data transfer time(TH+TL=1.25µs±600ns)

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	

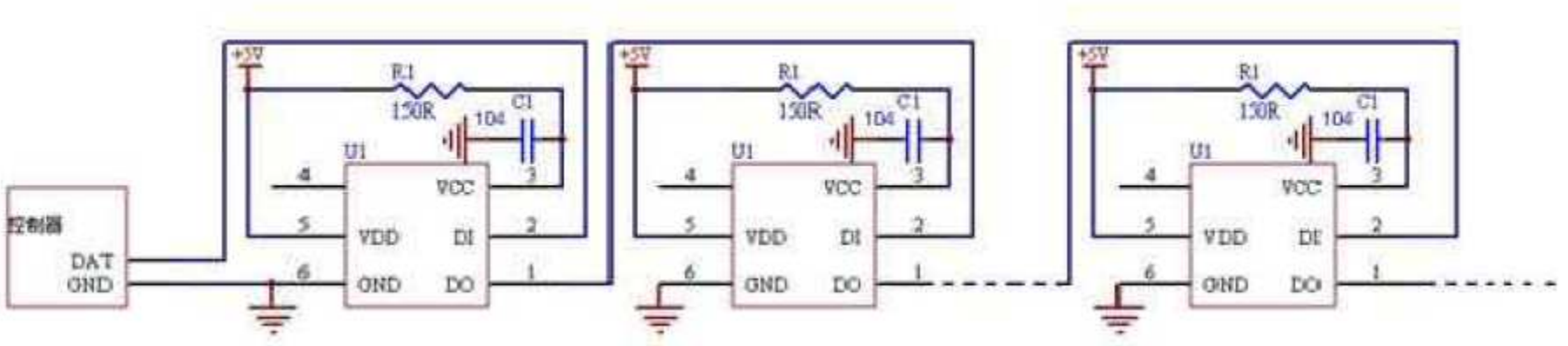
Sequence chart:



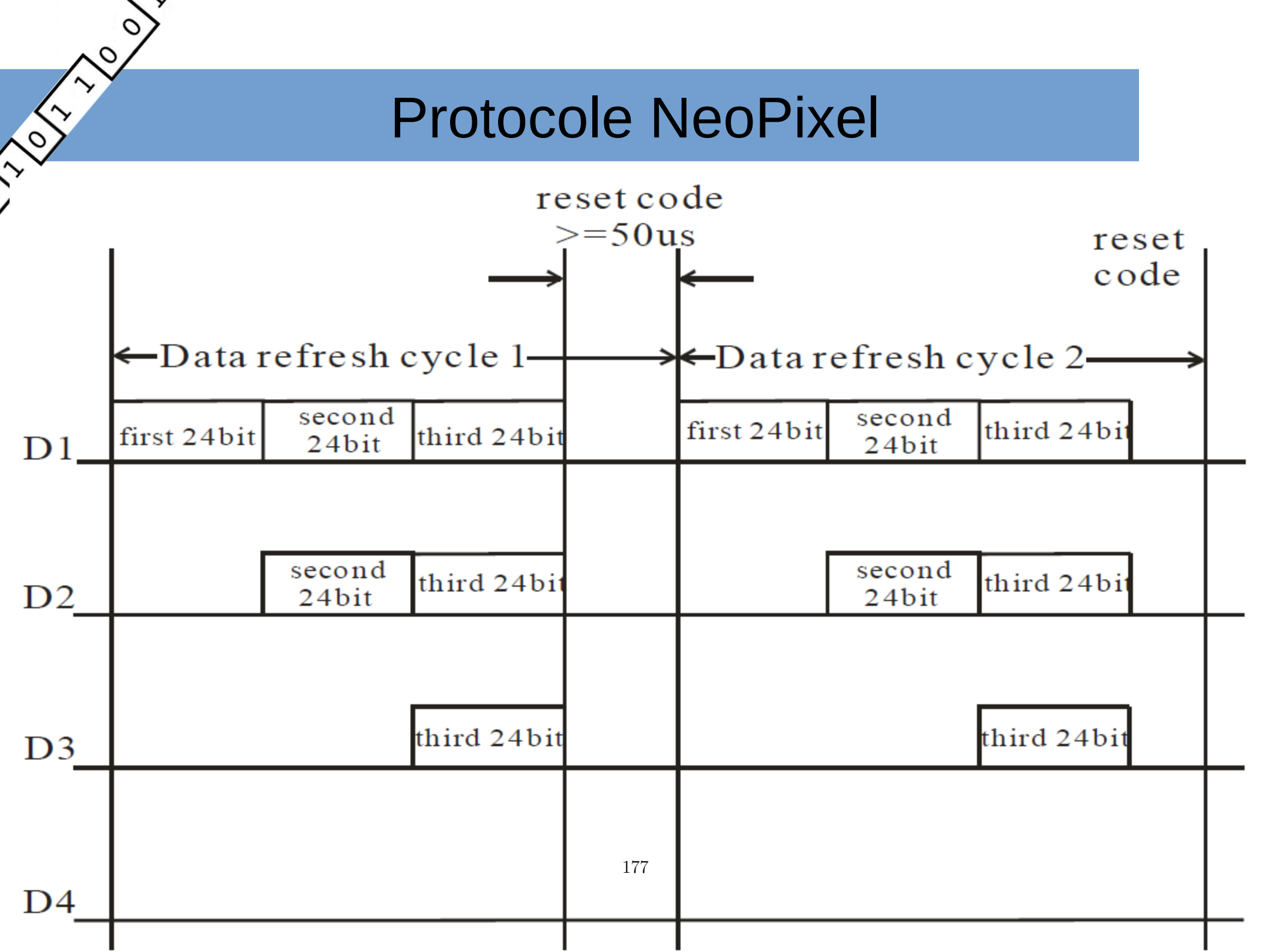
Protocole NeoPixel



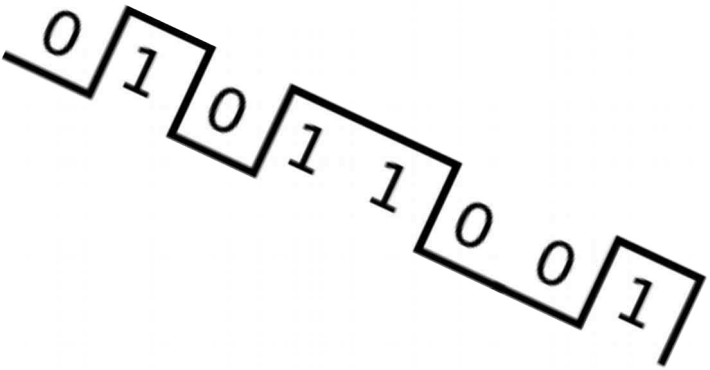
Protocole NeoPixel

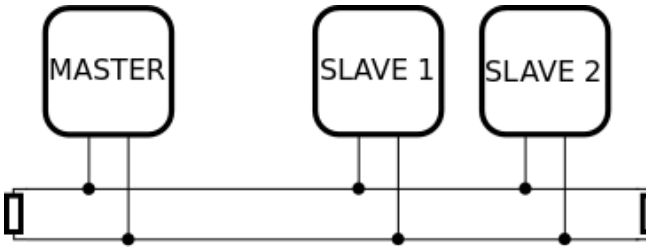


Protocole NeoPixel



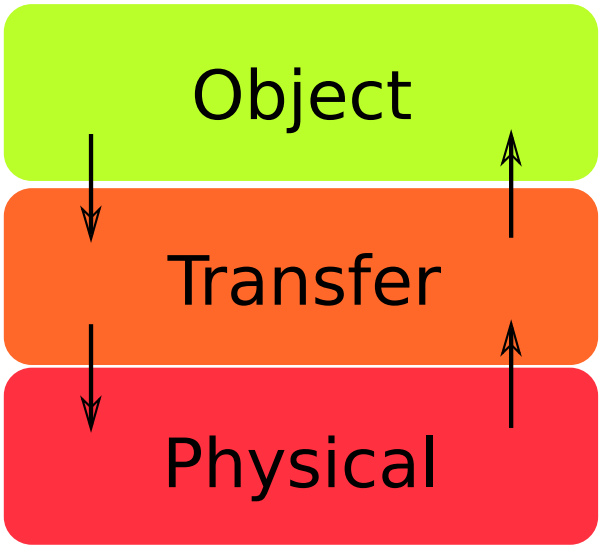
Bus CAN (Controller Area Network)





Bus CAN

- Carte d'identité:
  - Bus série asynchrone
  - Topologies supportées:
    - Point → Multi-points (BUS), Pair à Pair
  - Half-duplex
  - Deux vitesses de communication High-speed et Low-speed
  - Deux formats de trames
    - CAN 2.0A
    - CAN 2.0B ou extended
  - Bus robuste aux erreurs (chaque nœud intégre une gestion des erreurs en emission et réception)



Bus CAN

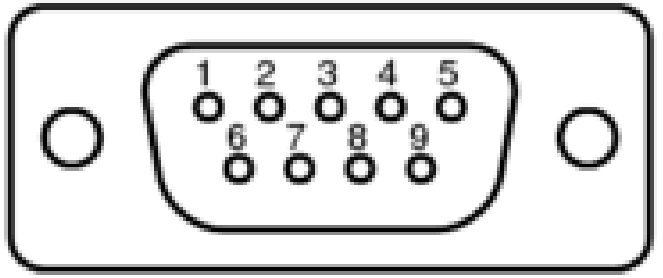
- Spécifications électriques:
  - Transmission différentielle
  - 0→ 2v tension différentielle
  - 1.5v → 3.5v tension mode commun
  - Etat dominant : 2v tension différentielle CAN_H 3.5v CAN_L 1.5v
  - Etat récessif : 0v tension différentielle CAN_H 2.5v CAN_L 2.5v
  - '0' codé par état dominant, '1' par état récessif
  - Alimentation des composants du bus
    - 7v → 30v pas normalisé
- Economie de cuivre !

Bus CAN

- Connecteur:
  - 4 conducteurs : CAN_H, CAN_L, GND, Vpower
  - Utilisation du connecteur SUB-D (9 pins)
  - Connecteur OBD pour les applications automobiles
- Câble à paire torsadée
- Jusqu'à 30 acteurs sur le bus
- 125 Kbit/s → 1 Mbit/s
- 500m → 40m

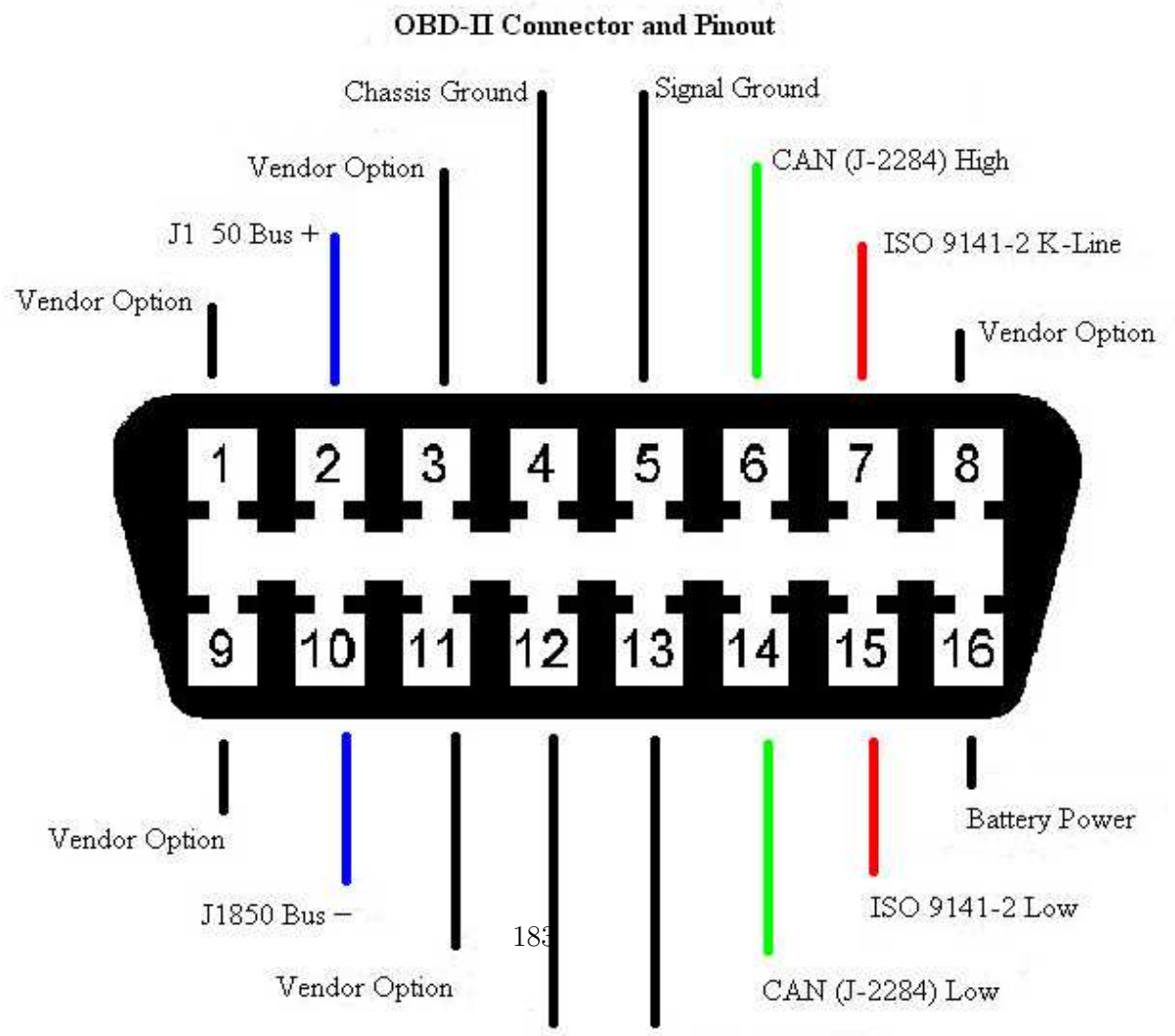
Bus CAN

- Connecteur DE-9
  - 1 (Réservé)
  - 2 CANL
  - 3 Masse
  - 4 (Réservé)
  - 5 Blindage (optionnel)
  - 6 Masse
  - 7 CANH
  - 8 (Réservé)
  - 9 Alimentation externe (optionnel)



Bus CAN

- Connecteur OBD



Bus CAN

- Couche Liaison/Transfert
  - Pas d'adressage direct du nœud de la communication
  - Méthode « d'abonnement » aux messages
    - Chaque nœud spécifie localement les identifiants de messages qu'il souhaite recevoir

Bus CAN

- Il y a quatre types de messages (Frames):
  - Data frame: Pour transmettre des données d'un nœud à un autre. Standard ou étendu.
  - Remote frame : message requête de données. Standard ou étendu.
  - Error Frame : Envoyé à tous les nœuds lorsqu'un nœud détecte une erreur sur le bus
  - Overload Frame : Permet de forcer un délai sur le bus (par exemple dans le cas d'un remote frame quand le périphérique ne peut fournir une réponse rapidement)

Bus CAN

- Format de message (bit MSB first)
  - Le début de trame ou SOF (Start Of Frame) matérialisé par 1 bit dominant,
  - Le champ d'arbitrage (identificateur) composé de 12 ou 30 bits,
  - Le champ de commande (ou de contrôle) composé de 6 bits,
  - Le champ de données composé de 0 à 64 bits (de 0 à 8 octets),
  - Le champ de CRC composé de 16 bits,
  - Le champ d'acquittement composé de 2 bits,
  - La fin de trame ou EOF (End of Frame) matérialisée par 7 bits récessifs.

Source wikipedia.fr

Bus CAN

- Champ d'arbitrage
  - Permet d'identifier le message (message id)
  - 11 bits dans CAN 2.0A, 29 bits dans CAN 2.0B
  - Terminé par 1 bit (RTR)
    - Dominant si data frame
    - Récessif si remote frame

Source wikipedia.fr

Bus CAN

- Champ de commande
  - 2 bits reservés (permet d'identifier si format standart ou étendu)
  - 4 bits Data Length Code (code la taille du paquet de données)

Source wikipedia.fr

Bus CAN

- Champ de données
  - Données du paquet (pas standardisé)

Source wikipedia.fr

Bus CAN

- CRC
  - Calculé à partir de l'ensemble des champs transmis jusque là (les bits de transparence ne sont pas pris en compte). L'ensemble constitue le polynôme f(x).
  - L'algorithme consiste dans un premier temps à multiplier f(x) par 2^15.
  - Ensuite le polynôme f(x) est divisé (modulo 2) par le polynôme g(x)=2^15+2^14+2^10+2^8+2^7+2^4+2^3+2^0.

Source wikipedia.fr

Bus CAN

- Utilisation du champ d'arbitrage
  - Permet de filtrer les messages reçus (un nœud ne s'abonne qu'a certains type de messages)
  - Filtrage matériel au niveau du récepteur
    - Mask : Masque de bit permettant les bits du champ d'arbitrage à contrôler
    - Filtre : Valeur à filtrer après masquage

Source wikipedia.fr

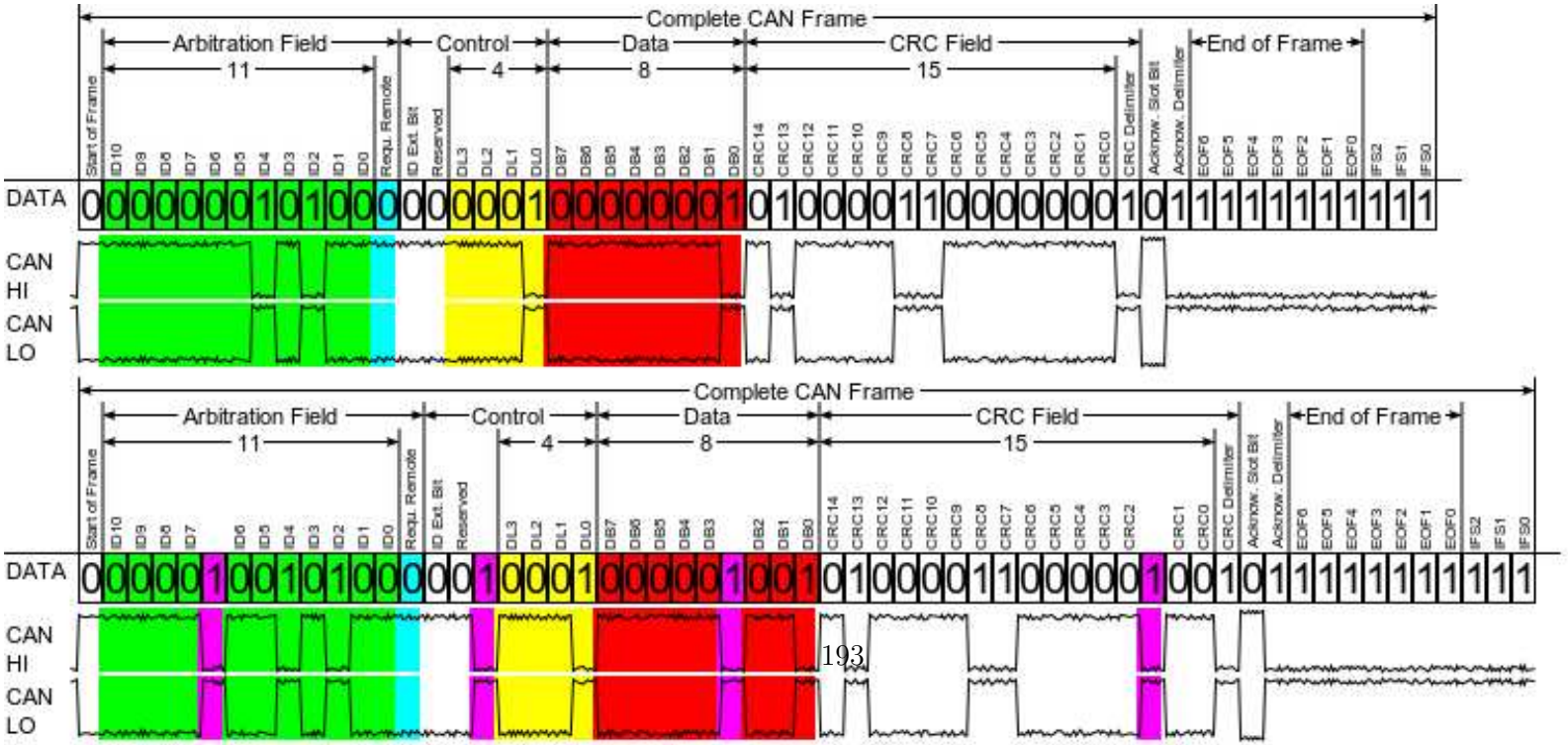
Bus CAN

- Utilisation du champ d'arbitrage
  - Exemple de filtrage

Source wikipedia.fr

Bus CAN

- Bit stuffing
  - Chaque séquence de 5bits de même valeur est suffixée par un bit de valeur opossée
  - Utilisée dans tous les champs sauf : CRC delimitter, ACK et EOF



Source wikipedia.fr

Bus CAN

- Priorité de transmission
  - Lorsqu'il transmet un bit, le nœud doit rester à l'écoute du bus.
  - Si il y une différence (forcément sur un bit récessif), c'est qu'un autre nœud l'a écrasé par un bit dominant
  - Le nœud interromp sa transmission, monitorer le bus pour attendre la fin de la transmission, puis tenter à nouveau d'envoyer son message.
  - La priorité est réalisée grâce au champ d'arbitrage.
    - Le champ adressage qui débute par la plus longue chaîne de '0' (état dominant) est le plus prioritaire
  - Phase de priorisation ou d'arbitrage prend fin au bit RTR.

Source wikipedia.fr

Bus CAN

- Priorité de transmission
  - Exemple de prise de priorité

Source wikipedia.fr

Bus CAN

- Trame d'erreur
  - 6 bits dominant ou récessifs (récessifs si l'emetteur et dans le mode erreur passive, dominants si dans le mode erreur active)
  - 8 bits recessifs

Bus CAN

- Types d'erreurs sur le bus
  - Bit error (erreur bit au niveau de l'emetteur après le RTR)
    - Avant le RTR, les collisions permettent de gérer la priorité
  - Stuff error (6 bits consecutifs avec la même valeur)
  - CRC error
  - CRC delimiter
  - ACKnowledge error
  - ACKnowledge delimiter

Bus CAN

- Recouvrement des erreurs
  - Par construction, la trame d'erreur brise la règle du bit-stuffing puisque l'on a quoiqu'il arrive les 6 bits du drapeau d'erreur sont identiques.

    1) Lorsqu'un nœud émet une trame d'erreur, tous les autres nœuds détectent donc une erreur de type « Stuff error » et se mettent à envoyer également une trame d'erreur.

    2) Le dernier nœud à émettre fournit le délimiteur (8 bits récessifs) et met fin à la cacophonie.

    3) Le nœud ayant émis la trame incriminée retente alors sa chance.

    4) Et ainsi de suite, jusqu'à ce que la trame passe ou qu'un de ses compteurs d'erreur fasse changer de mode d'erreur au nœud.

Bus CAN

- Gestion des erreurs sur le bus CAN
- Chaque nœud a 2 compteurs d’erreurs
  - transmission sur les trames émises (TEC transmit error counter)
  - sur les trames reçues (REC – receive error counter)

Bus CAN

- TEC
  - Incrémenté de 8 :
    - si le nœud envoie un drapeau d'erreur sauf :
      - En cas de « ACKnowledge error » et qu'il est en mode d'erreur passive,
      - En cas de « Stuff error » durant la période d'arbitrage,
    - Si le nœud reçoit 7 bits dominants consécutifs après un drapeau d'erreur ou un drapeau de surcharge
  - Décrémenté de 1 si la transmission du nœud se déroule sans erreur.

Bus CAN

- REC
  - Incrémenté de 1 si le nœud détecte une erreur de réception, sauf en cas de « Bit error » pendant un drapeau d'erreur ou un drapeau de surcharge,
  - Incrémenté de 8 :
    - si le nœud reçoit un bit dominant juste après un drapeau d'erreur,
    - si le nœud détecte un « Bit error » pendant un drapeau d'erreur ou un drapeau de surcharge,
    - si le nœud reçoit 7 bits dominants consécutifs après un drapeau d'erreur ou un drapeau de surcharge,
  - Décrémenté de 1 s'il est inférieur à 127 et que le nœud reçoit une trame sans erreur,
  - Ramené entre 119 et 127, s'il était supérieur à 127 et que le nœud reçoit une trame sans erreur.

Bus CAN

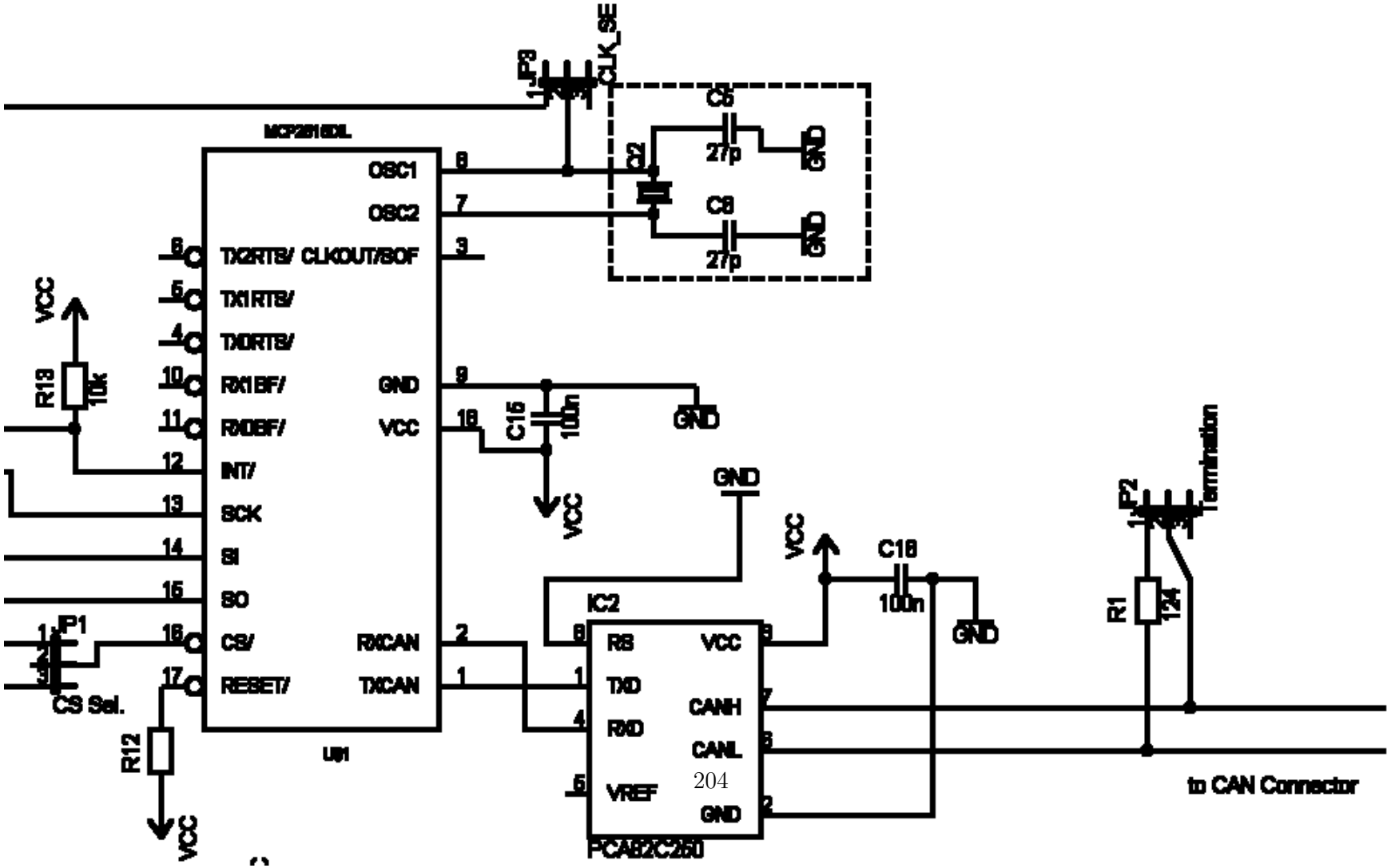
- Gestion des erreurs: 3 états...
- Selon la valeur des compteurs :
  - Etat Erreur-active : fonctionnement normal
  - Etat Erreur-passive : émission possible mais 8 bits après que le bus soit libre
  - Etat Bus-off : la station se déconnecte du bus (plus d’émission ni de réception)

Bus CAN

- Intégration:
  - Transceiver (PHY) CAN (MCP2551) : gère l'adaptation de niveau.
  - Contrôleur CAN (MAC) externe (MCP2515) : gère le filtrage des messages, la mise en forme des trames, la détection des erreurs. Nécessite d'ajouter un transceiver.
  - Contrôleur CAN intégré (NXP, Freescale, Microchip, Atmel ...) : Nécéssite d'ajouter un transceiver.

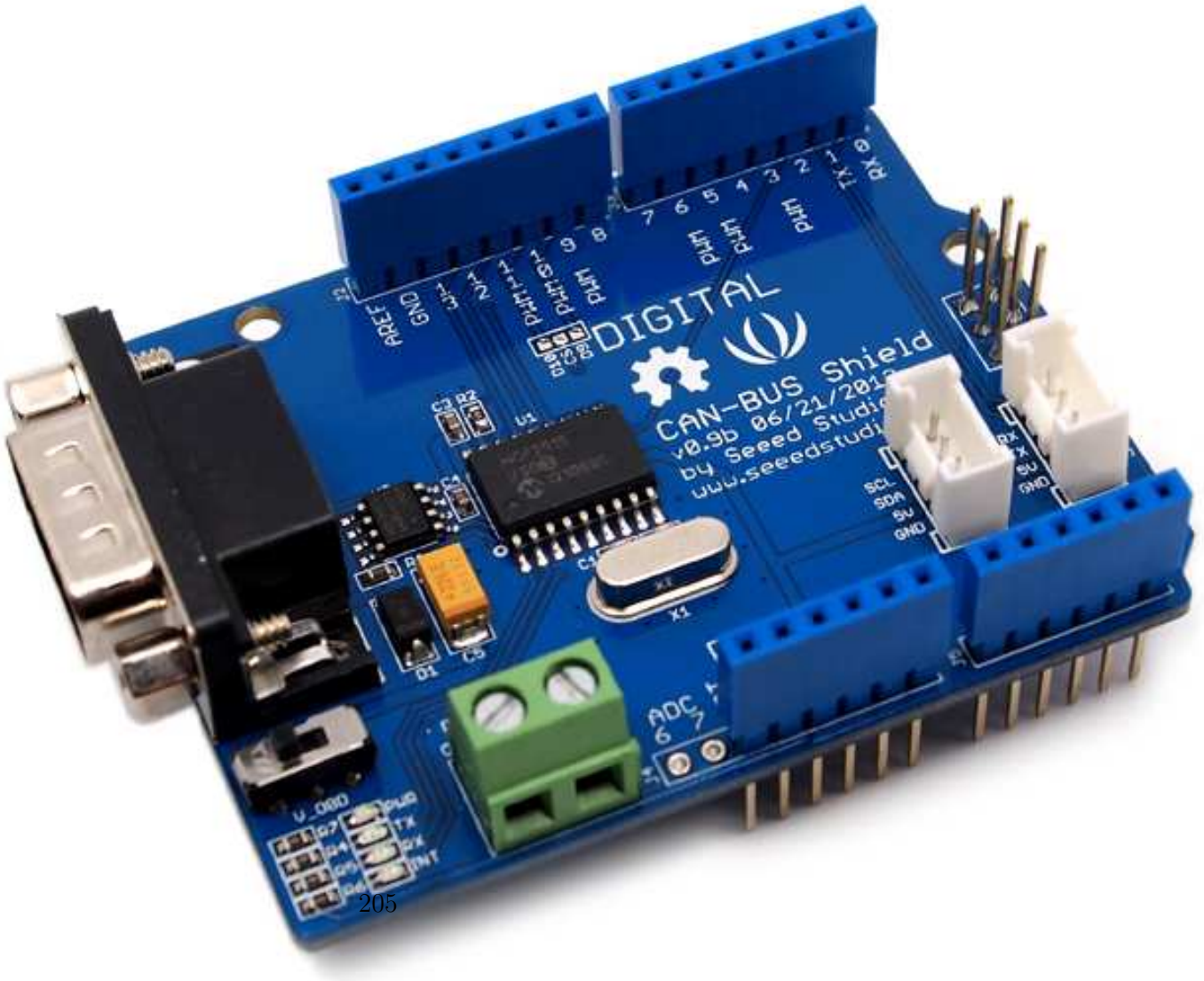
Bus CAN

- Intégration



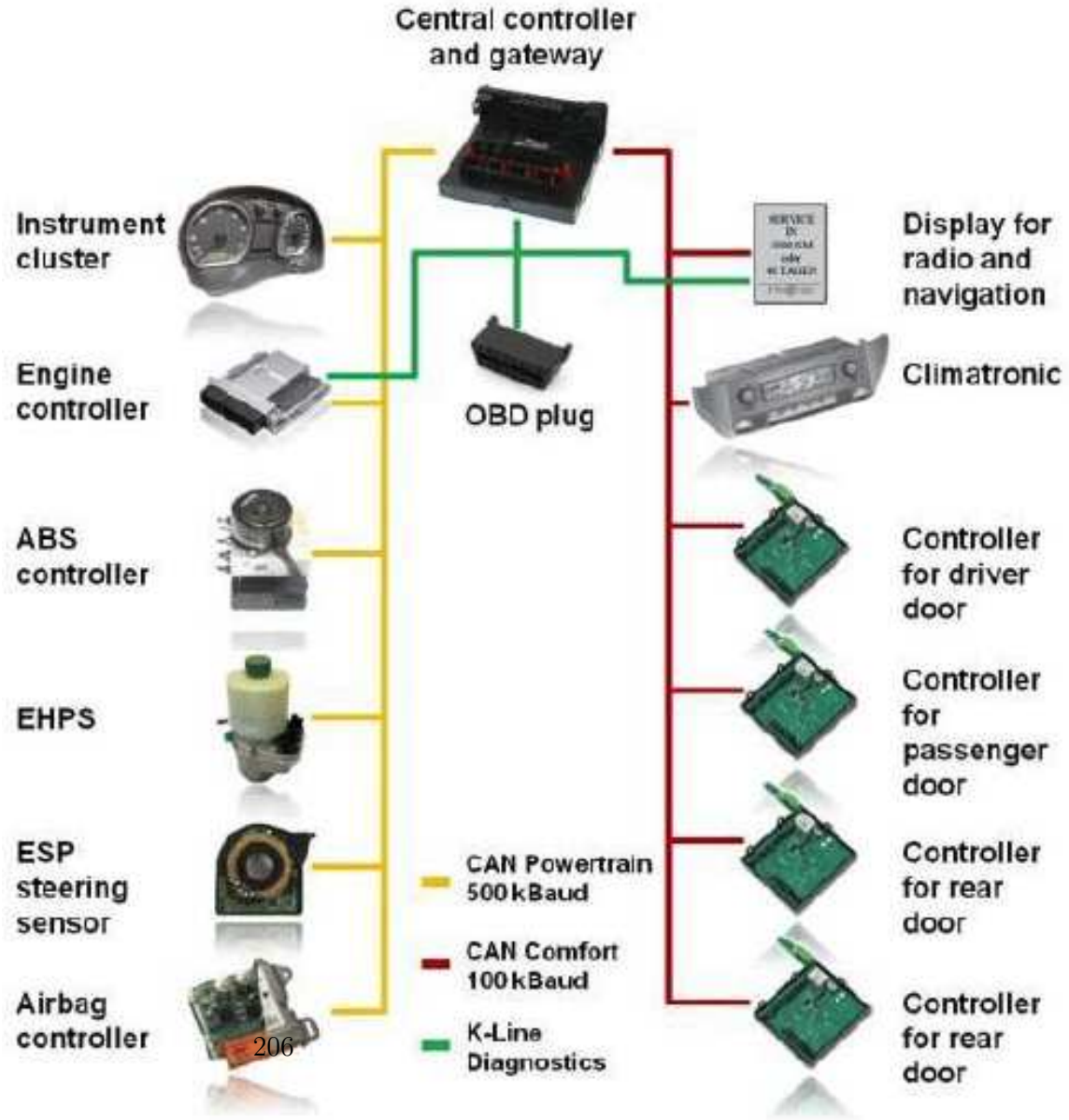
Bus CAN

- Arduino

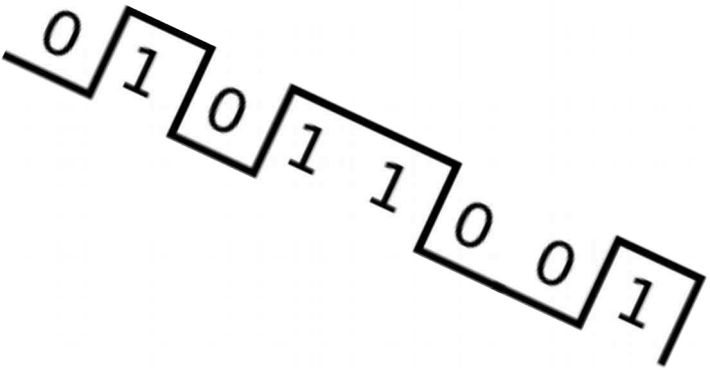


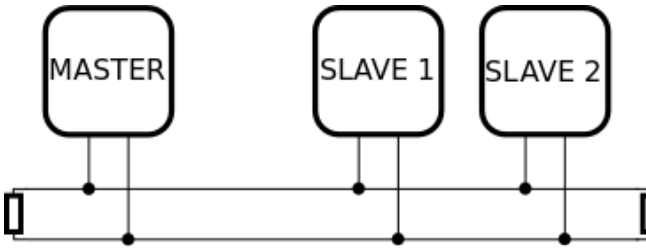
Bus CAN

- Automobile



Wire Less: Sans les fils
Modem 433Mhz
ZigBEE
Bluetooth
WIFI
GSM/GPRS





Sans les fils !

- Transmission de données sans fils
- Peut remplacer de manière transparente une liaison filiaire
- Problèmes inhérents:
  - Consommation
  - Immunité au bruit
  - Media partagé … (le champ éléctro-magnétique appartient à tous)
  - Sécurité

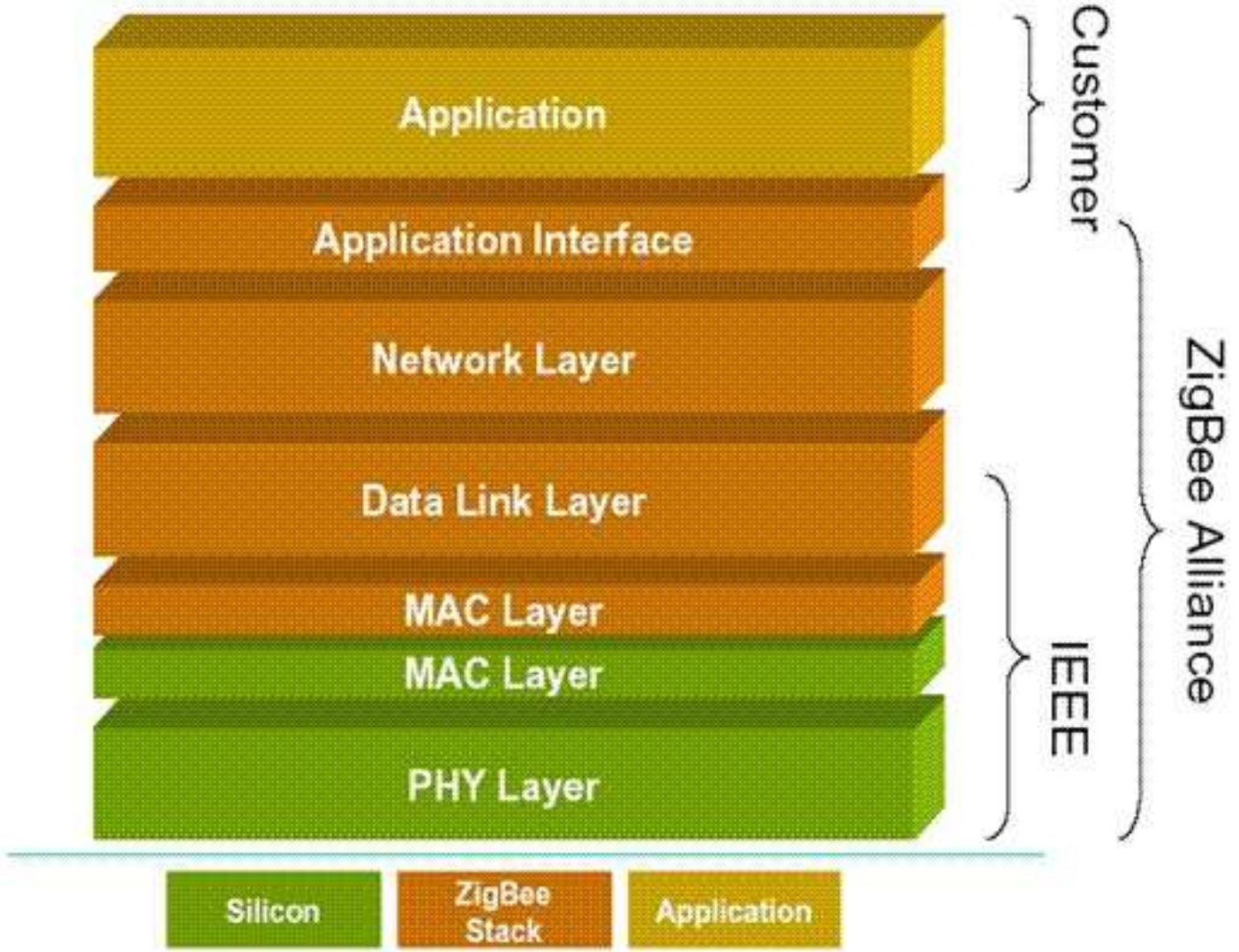
Liaison série 433Mhz

- Définit une couche physique de substitution pour une liaison série asynchrone
- Codage de canal : ASK, FSK, GFSK
- Vitesse : ~9600 bps
- Distance : > 500m
- Puissance : importante
- Sécurité : pas de cryptage établi

Liason série 433Mhz

- Applications:
  - Télécommande (portail, alarme)
  - Domotique

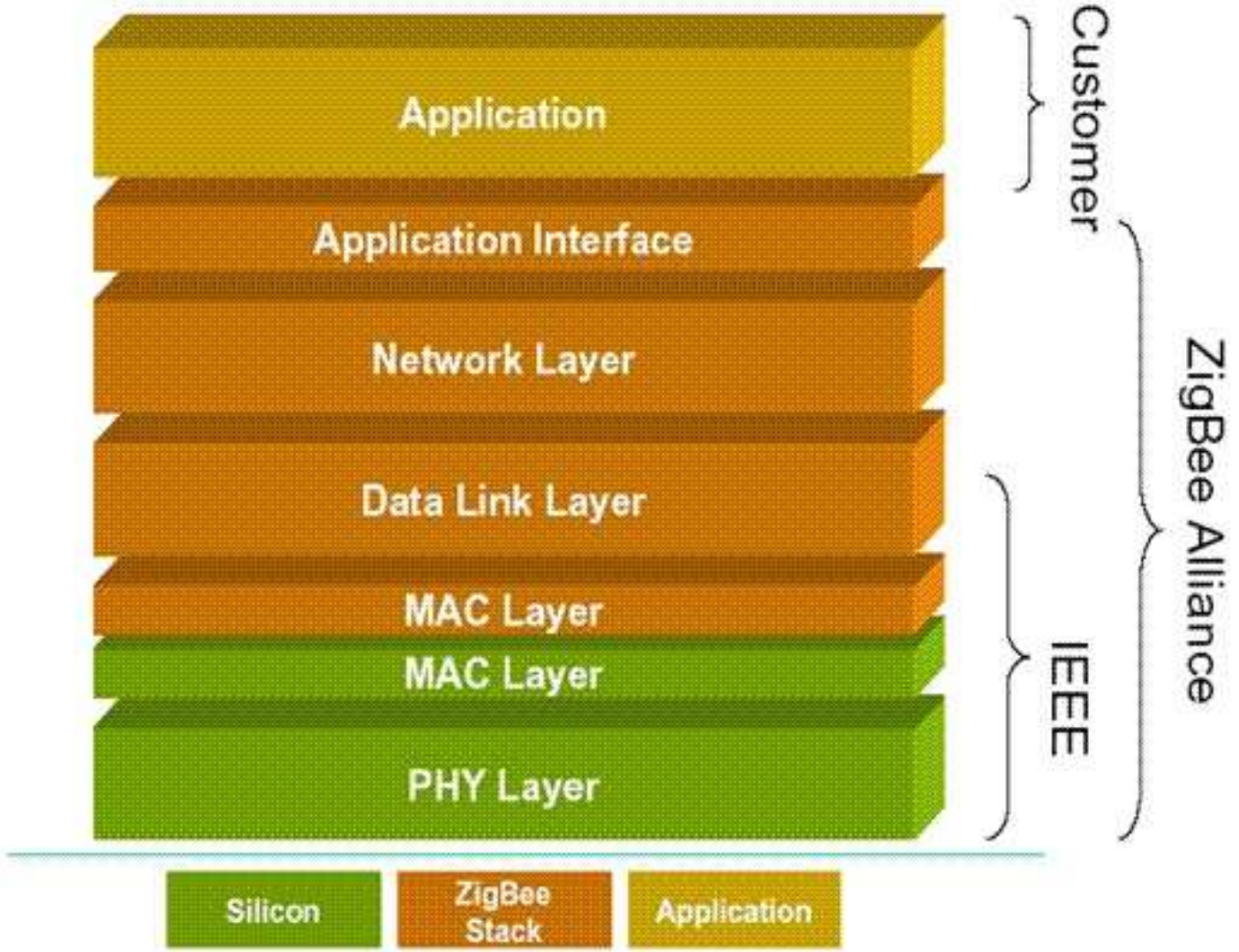
ZigBee


Application
Application Interface
Network Layer
Data Link Layer
MAC Layer
MAC Layer
PHY Layer
Customer
IEEE
ZigBee Alliance
Silicon
ZigBee Stack
Application

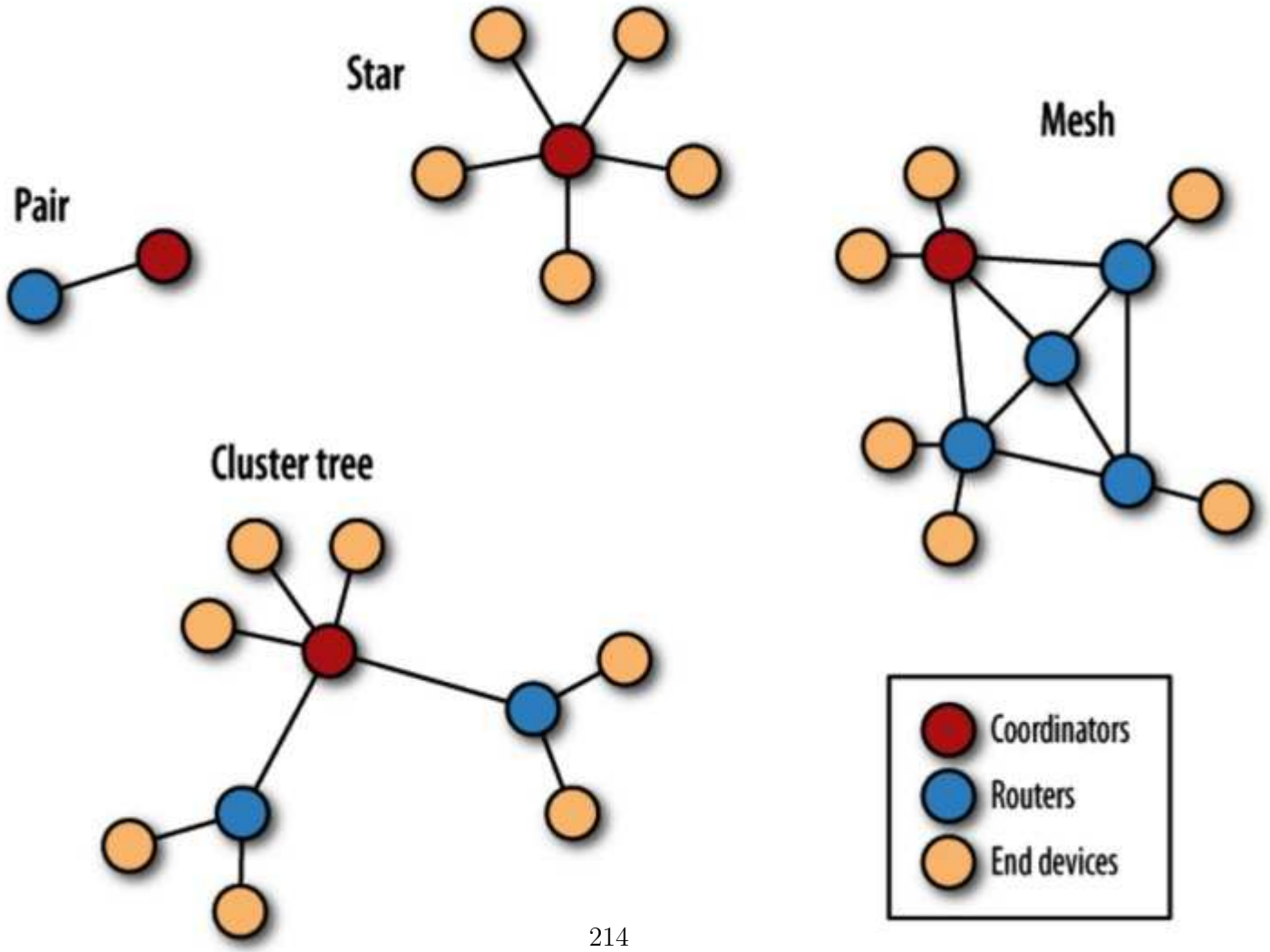
ZigBee

- Basé sur la norme radio 802.15.4
- Point → multi-points en maîtres-esclaves
- Basse consommation
- Vittesse : jusqu'à 250Kb/s
- Distance : jusqu'à 100m
- Consommation : ~40ma rx/tx, <10µa sleep)
- Sécurité : clé 128 bits
- Modulation :

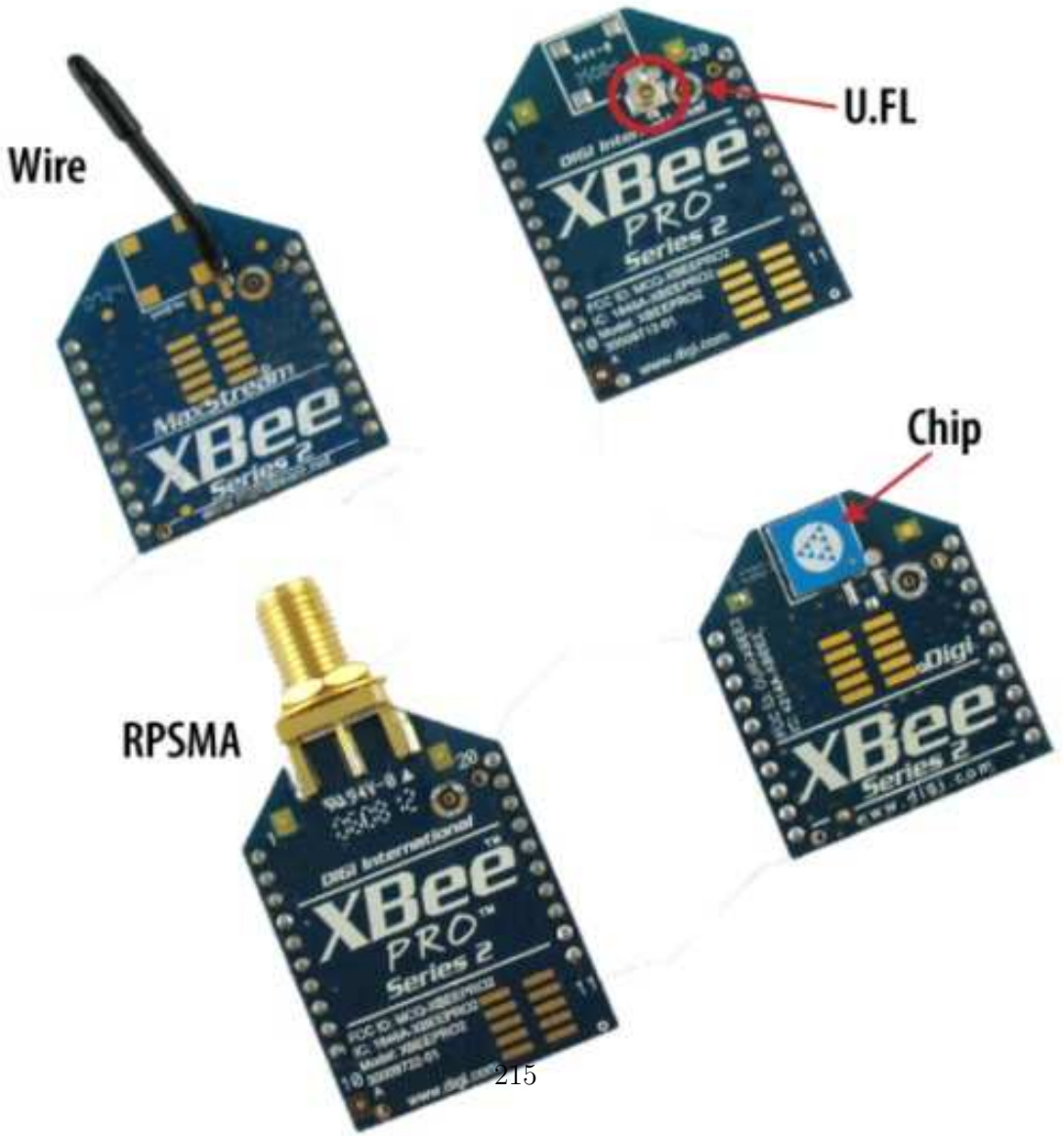
ZigBee


Application
Application Interface
Network Layer
Data Link Layer
MAC Layer
MAC Layer
PHY Layer
Customer
IEEE
ZigBee Alliance
Silicon
ZigBee Stack
Application

ZigBee

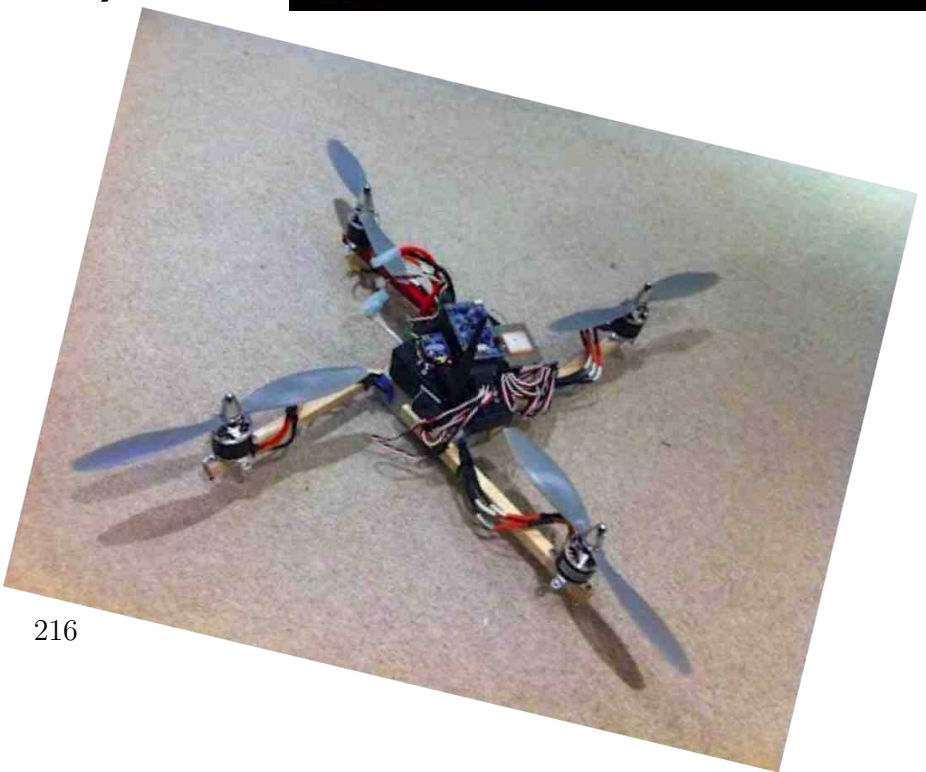

Star
Pair
Mesh
Cluster tree
Coordinators
Routers
End devices

ZigBee


Wire
U.FL
Chip
RPSMA
MaxStream
XBee
Series 2
XBee
PRO
Series 2
Digi
XBee
Series 2
Digi International
XBee
PRO
Series 2

ZigBee

- Applications:
  - Domotique
  - Réseau de capteurs
  - Internet Of Things (limité)
  - Télémétrie

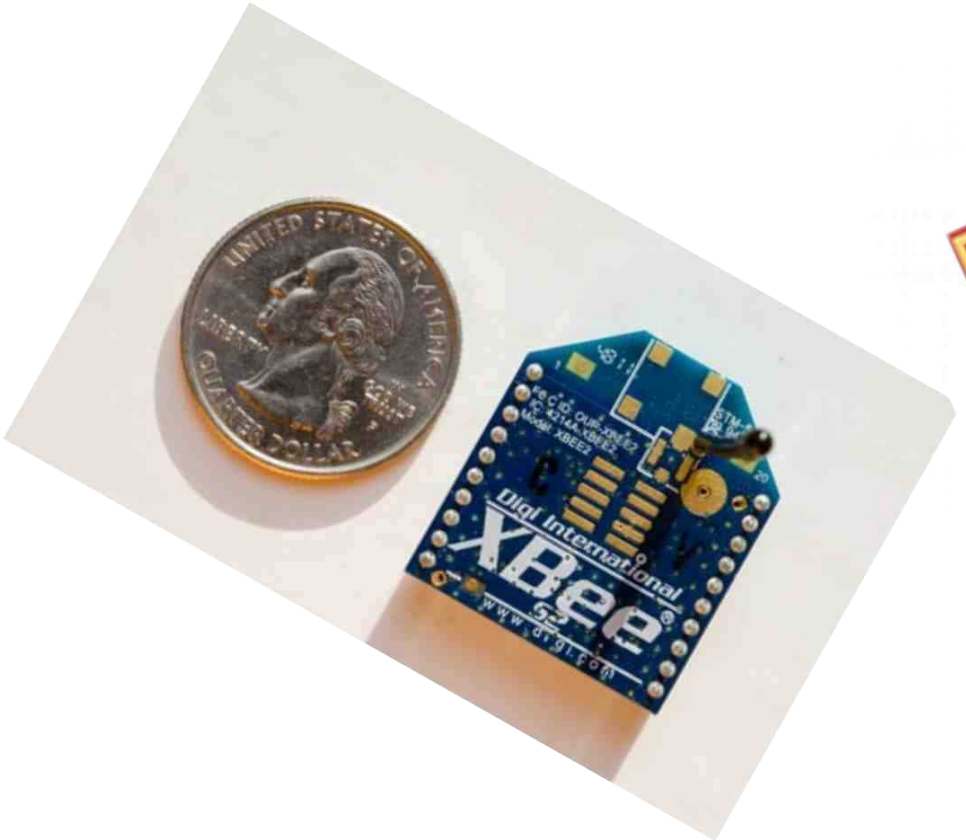


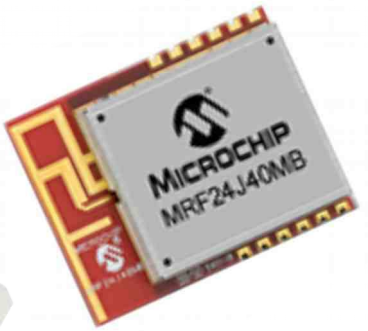
ZigBee

- Economie d'énergie
  - Utiliser de faibles vitesses de transmission
  - Transmettre le plus rarement possible
  - Dans le cas d'une alimentation solaire, transmettre quand l'energie le permet

ZigBee

- Intégration
  - Module Xbee (3.3v, série + IO + ADC)
  - Intégré sur micro-controlleur (Microchip, TI)





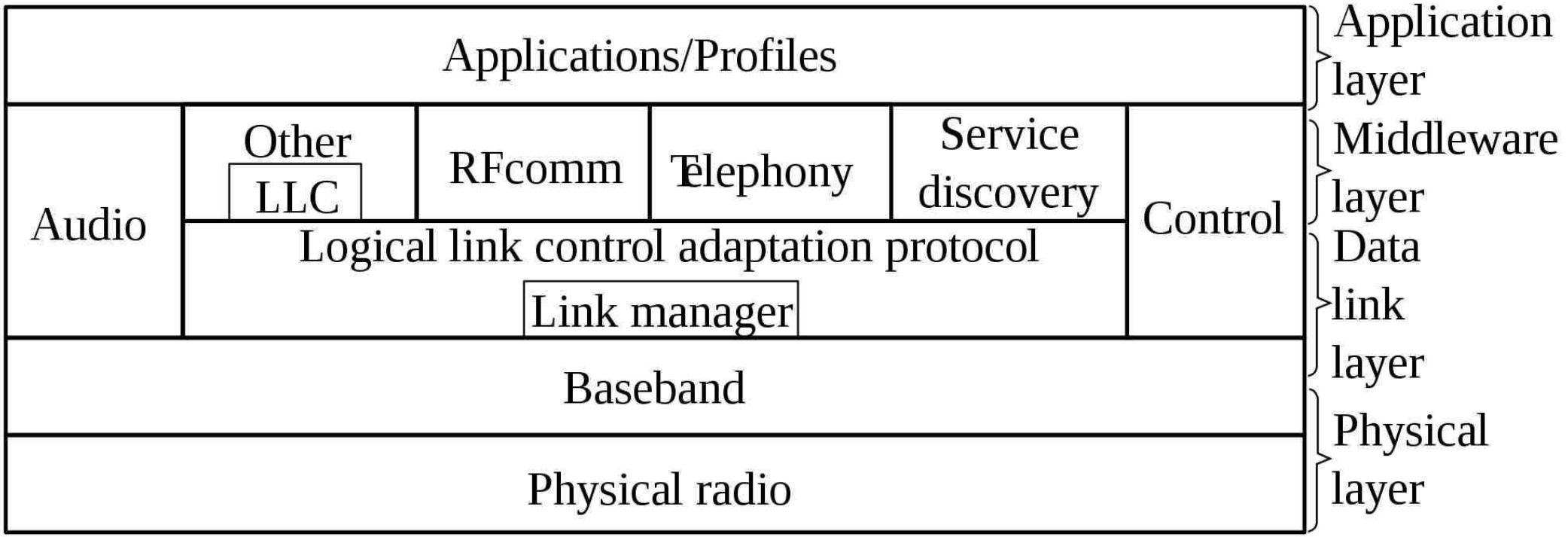
Bluetooth

- Basé sur la norme radio 802.15.1
- Point → multi-points en maître-esclaves (jusqu'a 7 esclaves)
- Vittesse : jusqu'à 24Mb/s
- Distance : jusqu'à 100m
- Consommation : 30ma tx/rx, 15ma BLE
- Modulation : GFSK, DQPSK, 8DPSK

Bluetooth

- Sécurité
  - Apairage par clé privée
  - AES-CCM

Bluetooth


Applications/Profiles
Audio
Other
LLC
RFcomm
Telephony
Service
discovery
Control
Logical link control adaptation protocol
Link manager
Baseband
Physical radio
Application
layer
Middleware
layer
Data
link
layer
Physical
layer

Bluetooth

- Services:
  - GAP : Generic Access Profile
  - SDAP : Service Discovery Application Profile
  - SPP : Serial Port Profile
  - HS Profile : Headset Profile
  - DUN Profile : Dial-up Networking Profile
  - LAN Access Profile : ce profil est maintenant obsolète ; il est remplacé par le profil PAN
  - Fax Profile
  - GOEP : Generic Object Exchange Profile
  - SP : Synchronization Profile
  - OPP : Object Push Profile
  - FTP : File Transfer Profile
  - CTP : Cordless Telephony Profile
  - IP : Intercom Profile
  - ...

Bluetooth

- Applications
  - Mobiles
  - Périphériques PC/Smartphone
  - Capteurs
  - Télécommande
  - Contrôle

Bluetooth

- Intégration
  - Module en liaison UART + IO + audio
  -

Wifi

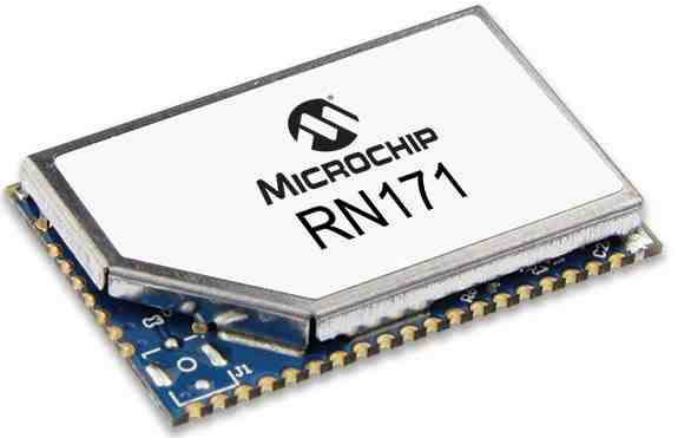
- Basé sur la norme radio 802.11
- Topologies:
  - Point → multi-points en pair à pair
- Forte consommation
- Vittesse : jusqu'à 6.5Gb/s (plus souvent ~54Mb/s)
- Distance : jusqu'à 250m
- Consommation : 120ma rx/tx, 50µa sleep
- Sécurité : WEP, WPA, ...

Wifi

- Applications
  - IOT (Internet Of Things)
  - Réseau de capteurs

Wifi

- Intégration
  - Module en liaison SPI (cc3000 TI, Microchip)



RN171 Wi-Fi® Module
(Part # RN171-I/RM)

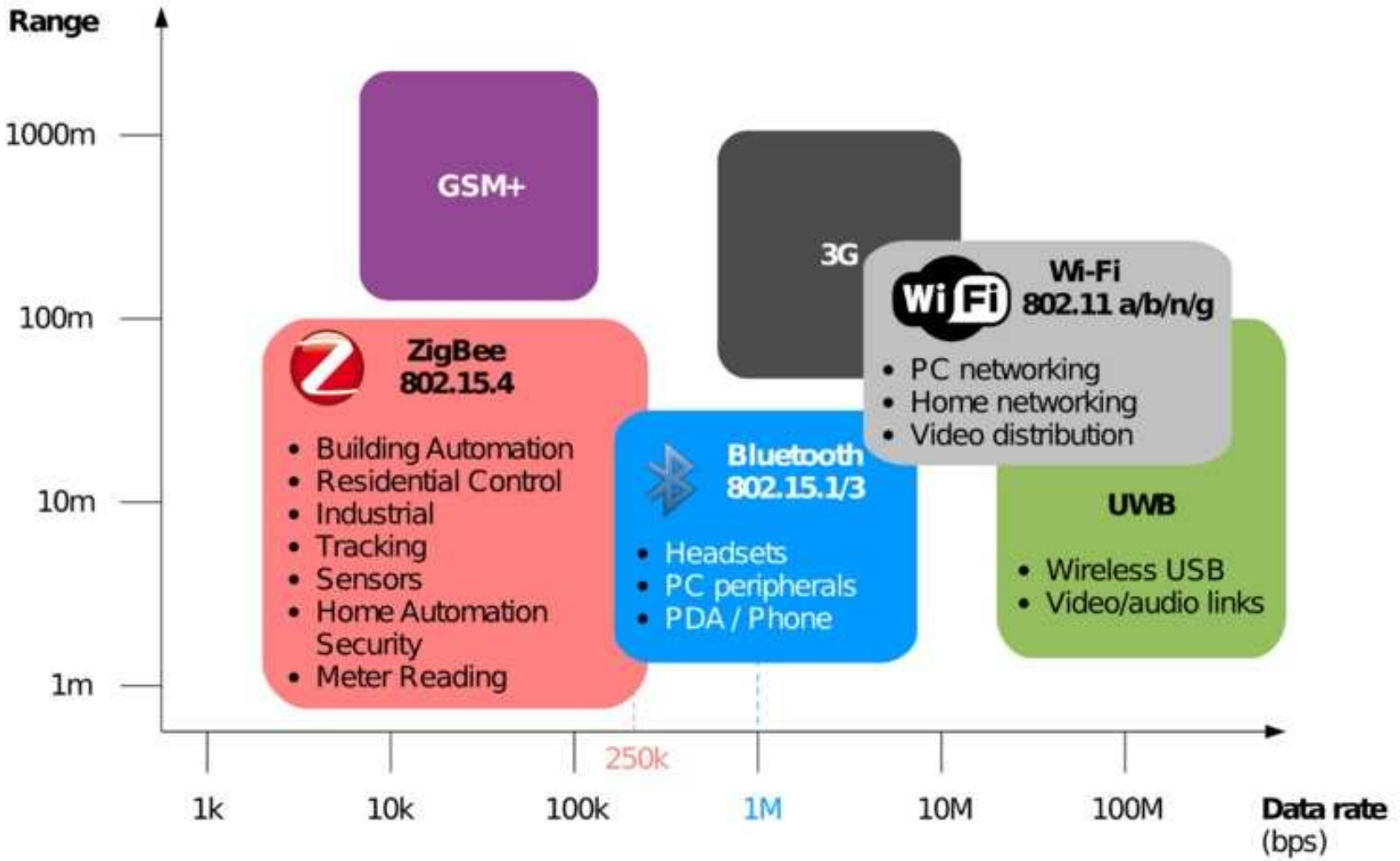
Wifi

- Intégration
  - Module prend en charge les couches basses (jusqu'à couche 4, Transport)
  - Décharge le micro-controlleur d'application

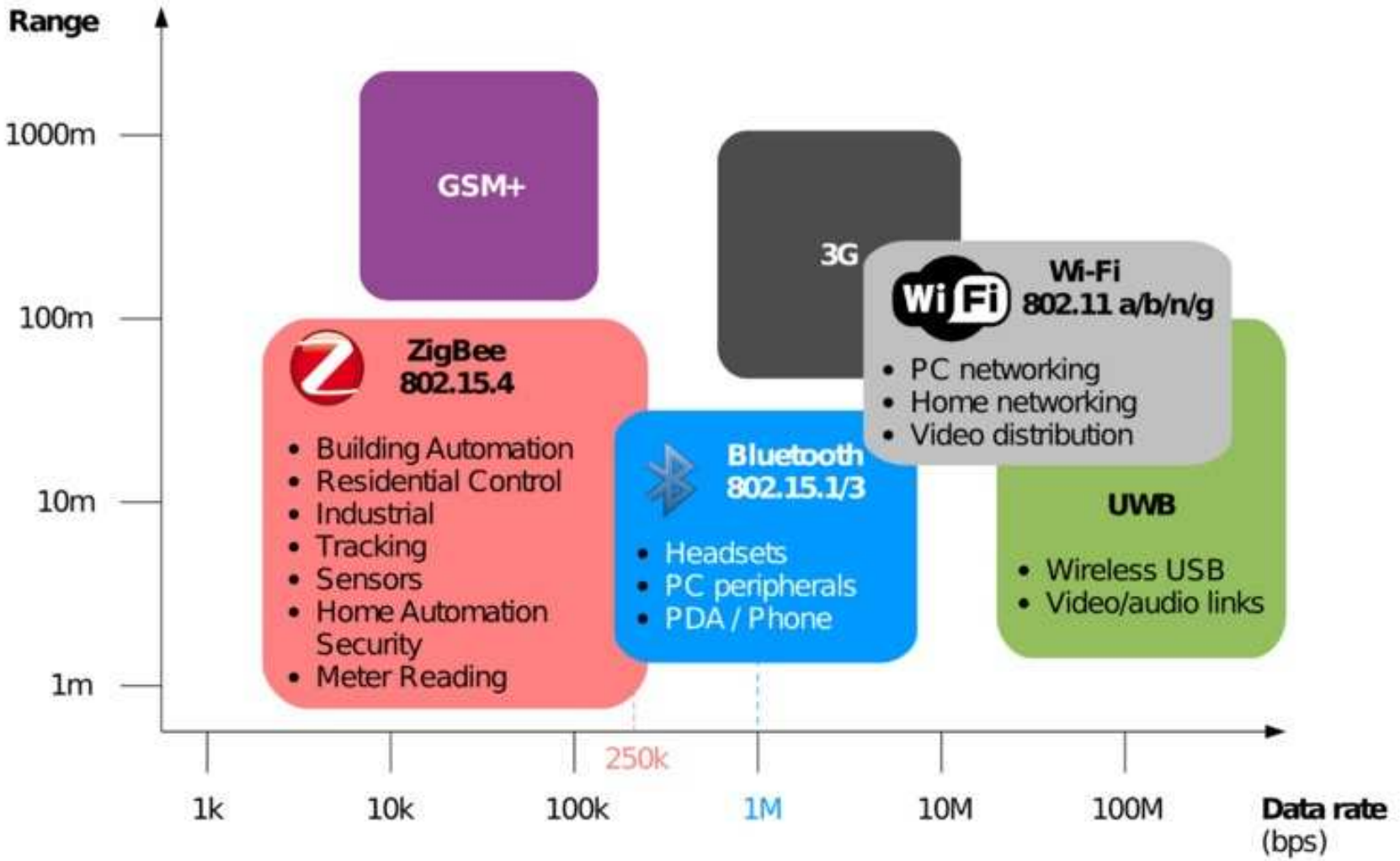
Zigbee vs Bluetooth vs Wifi

Caractéristique	Zigbee	Bluetooth	Wi-Fi
IEEE	802.15.4	802.15.1	802.11a/b/g/n/n-draft
Besoins mémoire	4-32 ko	250 ko +	1 Mo +
Autonomie avec pile	Années	Mois	Heures
Nombre de nœuds	65 000+	7	32
Vitesse de transfert	250 kb/s	1 Mb/s	11-54-108-320-1000 Mb/s
Portée (environ)	100 m	10 m	300 m

ZigBee


Range
1000m
100m
10m
1m
GSM+
3G
Wi-Fi
802.11 a/b/n/g
Wi Fi
• PC networking
• Home networking
• Video distribution
ZigBee
802.15.4
• Building Automation
• Residential Control
• Industrial
• Tracking
• Sensors
• Home Automation
Security
• Meter Reading
Bluetooth
802.15.1/3
• Headsets
• PC peripherals
• PDA / Phone
UWB
• Wireless USB
• Video/audio links
250k
1k
10k
100k
1M
10M
100M
Data rate
(bps)

ZigBee


Range
1000m
100m
10m
1m
GSM+
3G
Wi-Fi
802.11 a/b/n/g
Wi Fi
• PC networking
• Home networking
• Video distribution
ZigBee
802.15.4
• Building Automation
• Residential Control
• Industrial
• Tracking
• Sensors
• Home Automation
Security
• Meter Reading
Bluetooth
802.15.1/3
• Headsets
• PC peripherals
• PDA / Phone
UWB
• Wireless USB
• Video/audio links
250k
1k
10k
100k
1M
10M
100M
Data rate
(bps)

GSM/GPRS

- GSM : permet l'envoi de messages en mode texte SMS ou les appels vocaux
- GPRS : permet l'envoi de données brutes et la connexion à internet
- Débit : jusqu'à 20kbits/s
- Tarifs : ~0.01 euros/10 Ko

GSM/GPRS

- Intégration
  - Module (sim900, Siemens MC55), liaison série
  - Recyclage d'un vieux téléphone, liaison série

Sub-Ghz

- PAN (Personnal Area Network)
- BAN (Body Area Network)