# ELEC344

# Robotique et Systèmes Embarqués

## Bus

Alexis POLTI

# Bus : généralités

## Bus

- définitions :
  - ensemble de fils (connexions) partageant un même rôle
  - plus généralement : moyen de communication entre deux (ou plusieurs) composants, composé d'éléments structurels (connexions) et sémantiques (protocole)

- exemple : bus parallèle
  - données
  - adresses
  - signaux de contrôle

- enjeux :
  - augmenter la bande passante
  - diminuer les coûts

# Bus : généralités

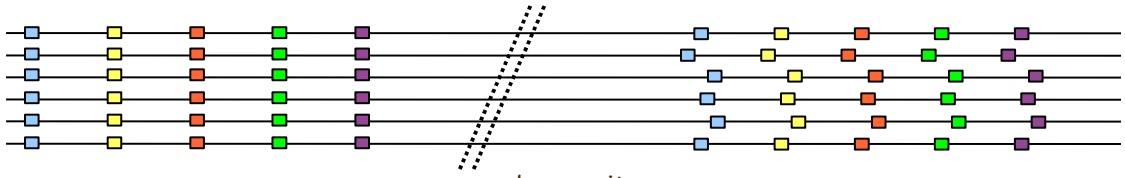
## Augmentation BP

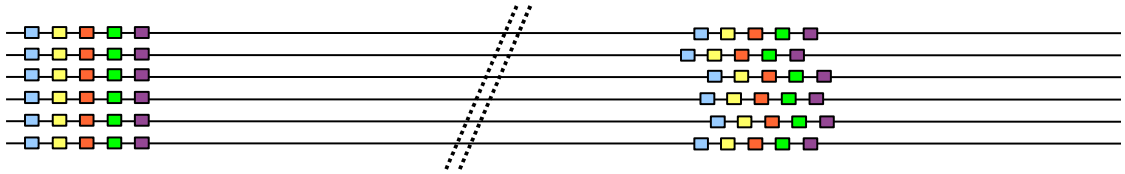
- structurel
  - largeur des bus
  - augmentation de la fréquence de fonctionnement
  - multidrop
- protocole
  - transferts démultiplexés
  - pipe-line
  - burst
  - overlapped arbitration / split phases
- conséquences :
  - bus parallèles synchrones linéaires non multiplexés
  - exemple : PCI
    - 32 / 64 bits
    - 49 lignes (32 bits)
    - 33 / 66MHz / 133MHz (PCI-X) / 266 MHz / 533 MHz

# Bus : généralités

## Inconvénients

- encombrement
  - routage
  - connecteurs
  - câbles nappe
- timing skew

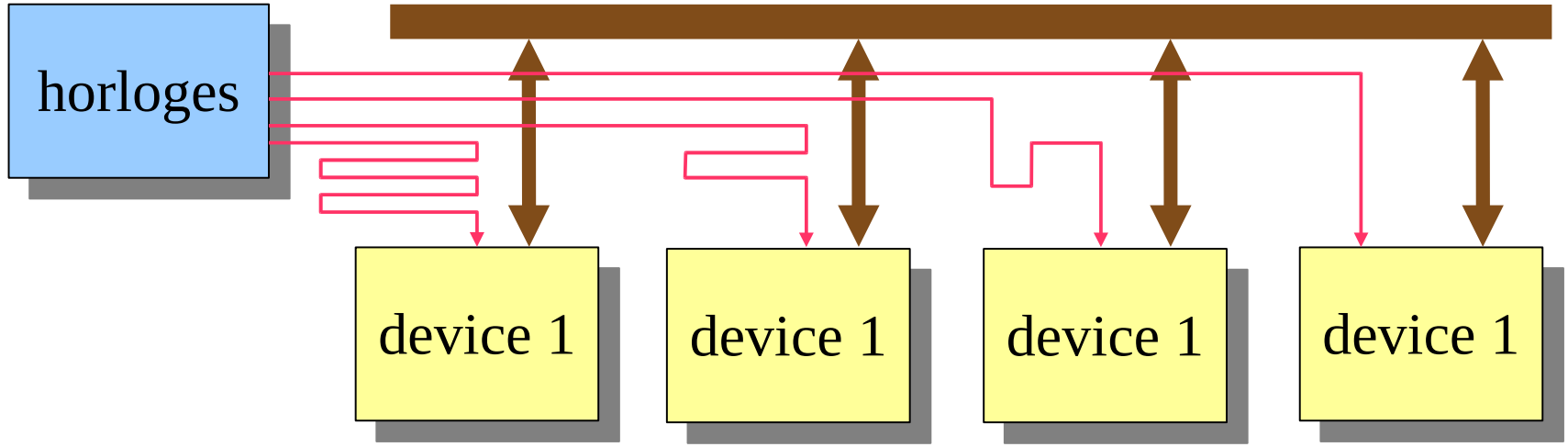



- fan out
- crosstalk
- consommation, ...

## Réduction du timing skew

- équilibrage des chemins

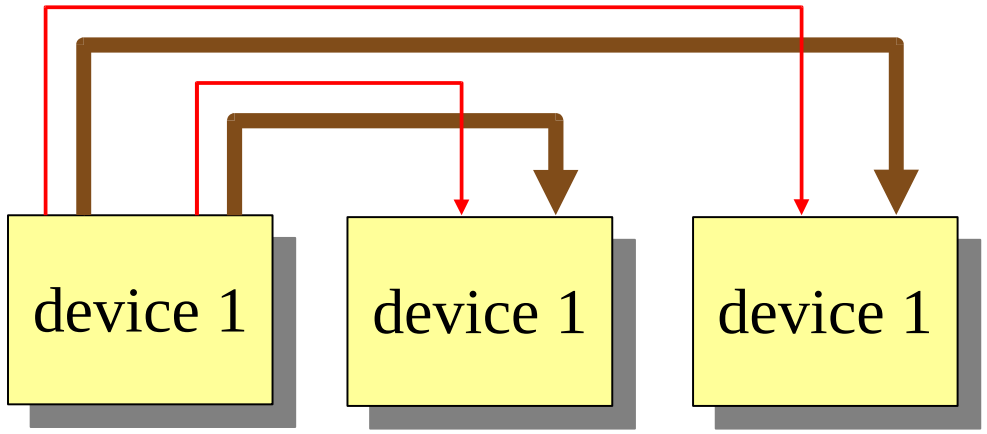



- difficultés de routage
- vitesse limitée par la longueur du chemin au dispositif le plus éloigné

# Bus : généralités

## Réduction du timing skew

- sources synchrones
  - chaque maître envoie sa propre horloge de la même façon que les données



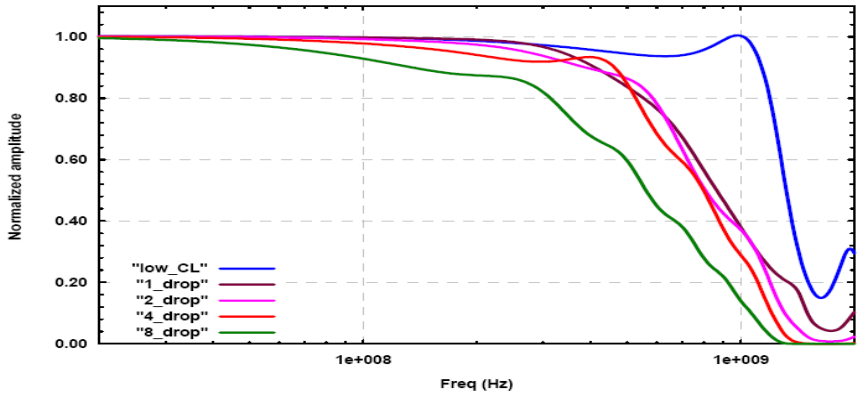

- problèmes :
  - nécessite une DLL dans chaque récepteur (quel déphasage ?)
  - complexité accrue
  - augmente le nombre de lignes d'horloge (une par paire émetteur / récepteur)

- exemple : DDR, FSB x86, HT, SPI-4.2 (Ethernet 10Gb)

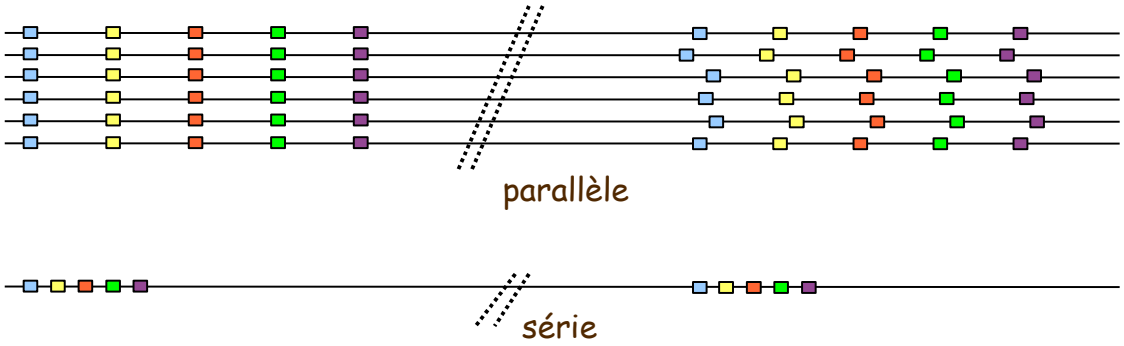
# Bus : généralités

## Augmentation BP

- point à point
  - atténuation !
  - cf. intégrité du signal




- bus série

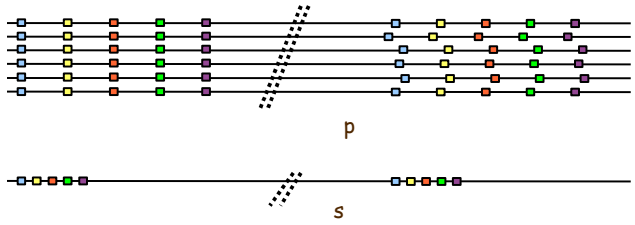



  - horloge séparée (bus synchrone)
  - horloge encodée dans les données (embedded clock, CDR)

# Bus : généralités

## Bus série

- sérialiseur / dé-sérialiseur (SERDES)
- égalisation
- problème de la latence...




- exemples :
  - serial-SCSI, SATA, PCI Express, IEEE1984
  - USB, Ethernet,
  - SPI, I2C, I2S, RS232, RS485, CAN, DMX512, DCC

# Bus : généralités

- performances / coût
  - bus hiérarchiques
- arbitration
  - simple / complexe
  - centralisée / distribuée
- beaucoup de bus sont dédiés à une application
  - SPDIF
  - SATA, Serial Attached SCSI
  - IIS
  - DMX512
  - DCC
  - ...

# Bus courants dans l'embarqué

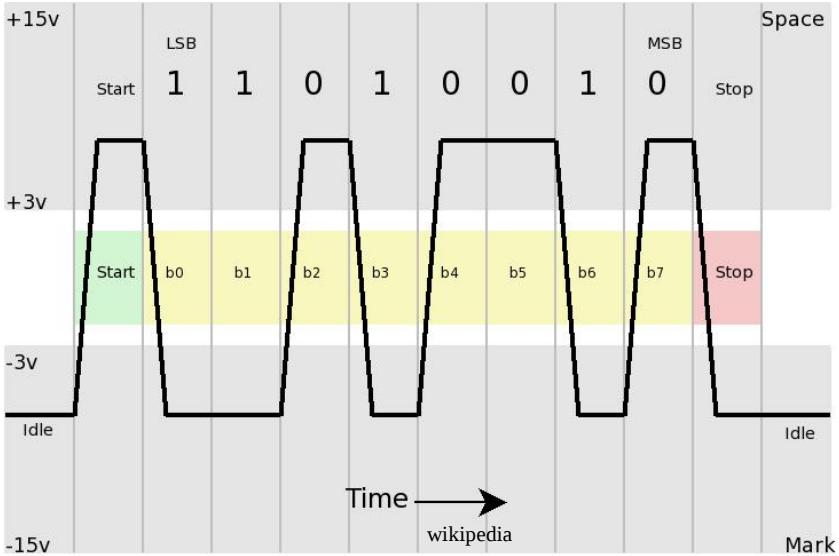
- RS232 :
  - bus lent (9600 – 115200 bps), bidirectionnel, point à point
  - usuel sur les vieux PC, rare mais trouvable dans les PC récents,
  - courant dans les systèmes embarqués
- RS422, RS485 :
  - différentiel, half-duplex, rapide, multidrop
- I2C :
  - bus lent (400kbps), multidrop, multi-maîtres, arbitration distribuée
  - adressage simple
  - variantes : SMBUS, DDC, TWI, ...
- CAN :
  - bus semi-rapide (1Mbps), multi-maîtres, arbitration distribuée automatique
  - pas d'adressage (mais message ID)
  - peu sensible au bruit
  - voir aussi : LIN, FlexRay

# Bus courants dans l'embarqué

- SPI :
  - bidirectionnel ( full-duplex), 10Mbps
  - un maître, plusieurs esclaves (mélange de multidrop et pt à pt, daisy chain possible)
  - extrêmement simple, utilisé pour flash, EEPROM, MMC / SD, Ethernet, son, LCD, capteurs, ...
  - variantes : I2S
- USB :
  - bus multi-vitesse (LS, FS, HS), différentiel, topologie arbre
  - 1 maître, plusieurs esclaves
  - courant dans les PC
- Firewire / IEEE1394 :
  - haut débit (100-800 Mbps), isochrone, topologie arbre
  - multi-maîtres, point à point
  - plus efficace que l'USB, mappé en mémoire

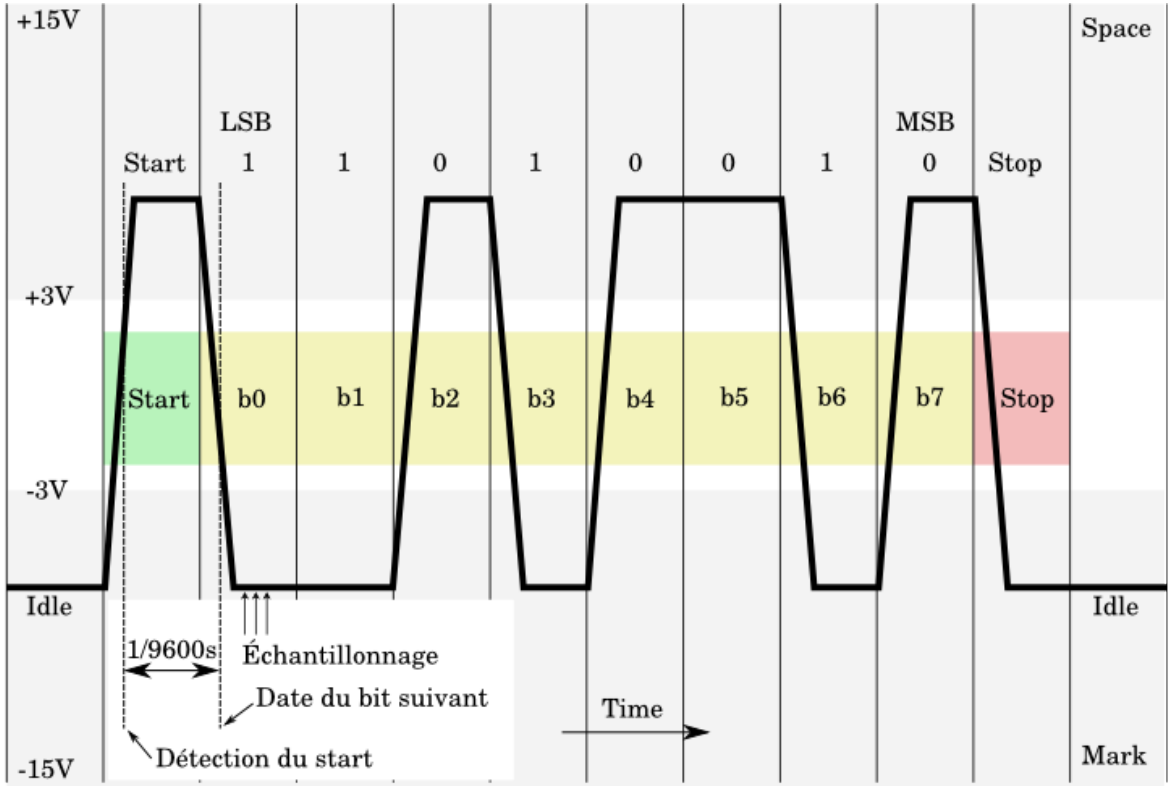
# Bus : RS232

- couramment appelé « port série »
- électriquement
  - asynchrone (9600, 19200, 38400, 57600, 115200 bps)
  - 1 (repos, mark) : -3 à -15V, 0 (actif, space) : +3 à +15V
  - 2 fils : RX / TX
- protocole
  - point à point, full duplex
  - contrôle de flux logiciel (échappement) ou matériel (fils supplémentaires)
  - format des données
    - data : 5, 6, 7, 8, 9 bits
    - parité : 0 ou 1 bit
    - stop : 1, 1.5, 2 bits



# Bus : RS232

- nécessité d'avoir des horloges synchronisées à la même vitesse
- plusieurs échantillonnages de chaque bit (généralement 3 ou 5) pour une réception fiable, centrés autour du milieu du bit
- tolérance d'un petit décalage
- détection possible des erreurs (*framing errors*)
- condition exceptionnelle (*break*)

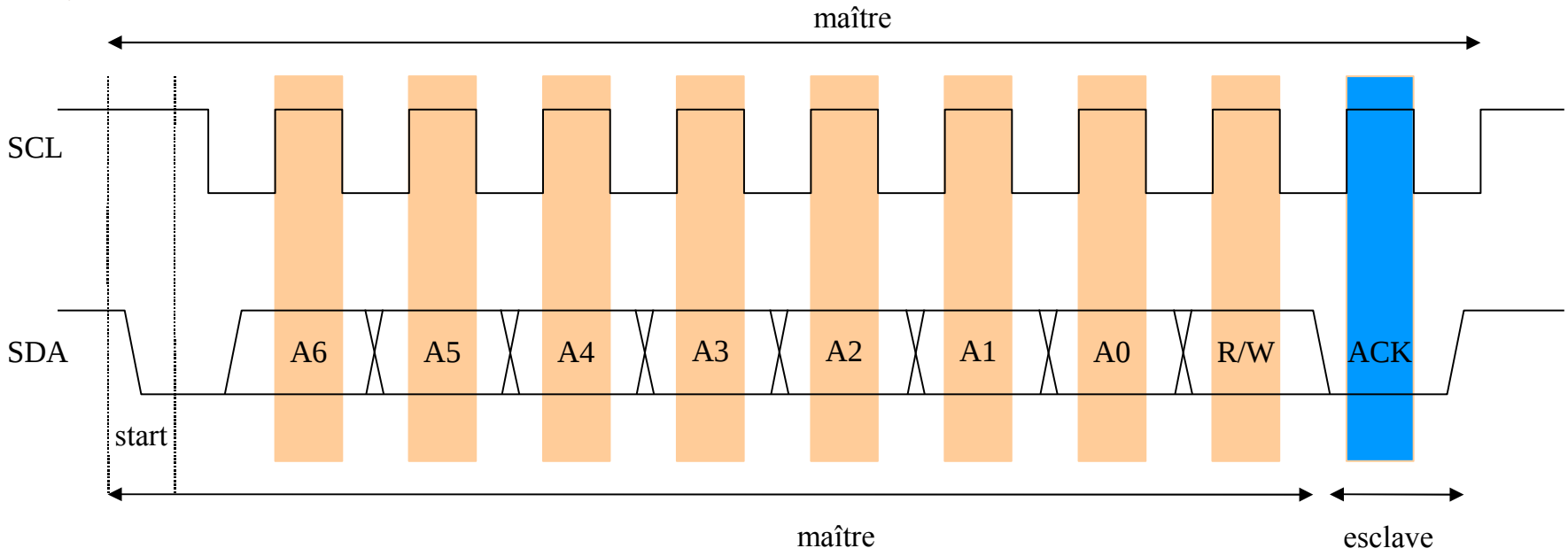


# Bus : I2C

- électriquement
  - 2 fils (SCL / SDA), collecteur ouvert (donc pull-up !)
  - 1 (repos) : 3.3 – 5V, 0 (actif) : 0V
- protocole
  - maître : impose l'horloge et transmet les adresses
    - transition sur SDA lorsque SCL = 0 : donnée
    - transition sur SDA lorsque SCL = 1 : start ou stop
  - clock stretching : un esclave trop lent peut maintenir SCL bas
  - pas de sémantique sur les messages (à part adresse 0 : « general call »)
  - message :
    - une phase d'adresse
    - une ou plusieurs phases de données
  - arbitration tout au long du dialogue

# Bus : I2C

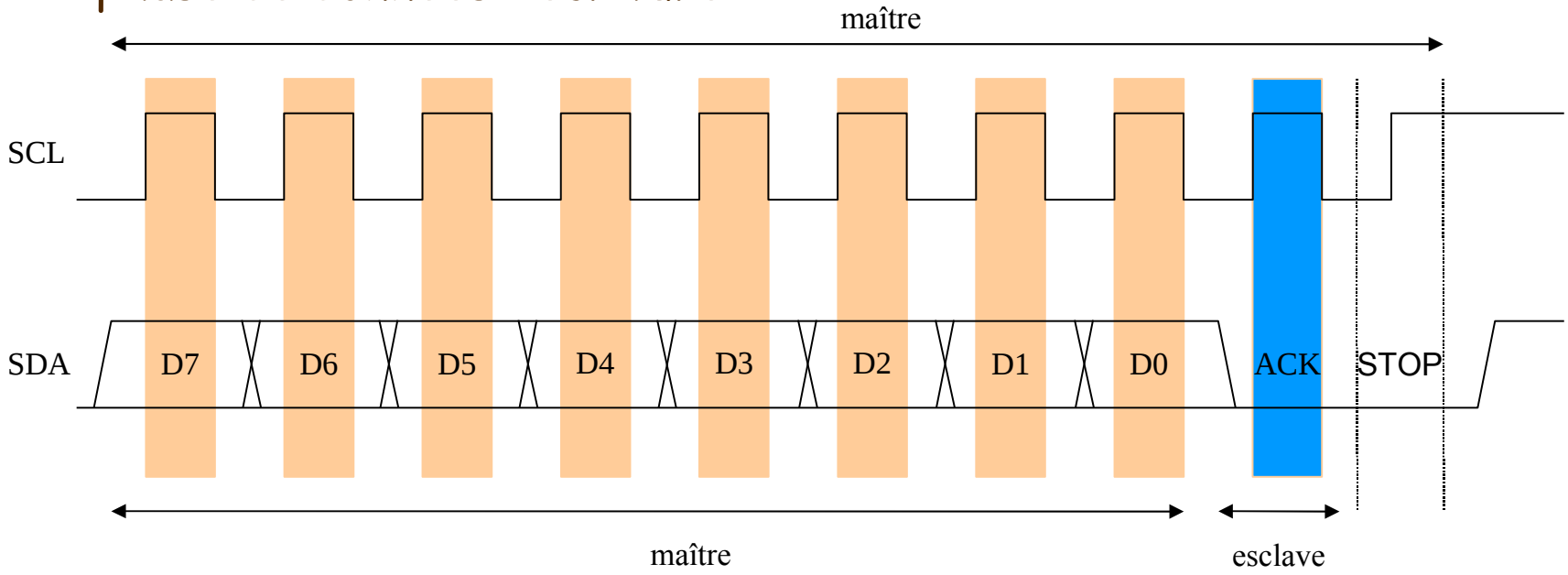
- phase d'adresse




- un esclave ayant reconnu son adresse répond par un ACK.
- un peu moins de 128 adresses disponibles
- adresses réservées :
  - 0000000 : general call (réglage adresses, SOS, ...)
  - 0000001 – 0000111, 111111xx : réservées
  - 11110xx : 10 bits

# Bus : I2C

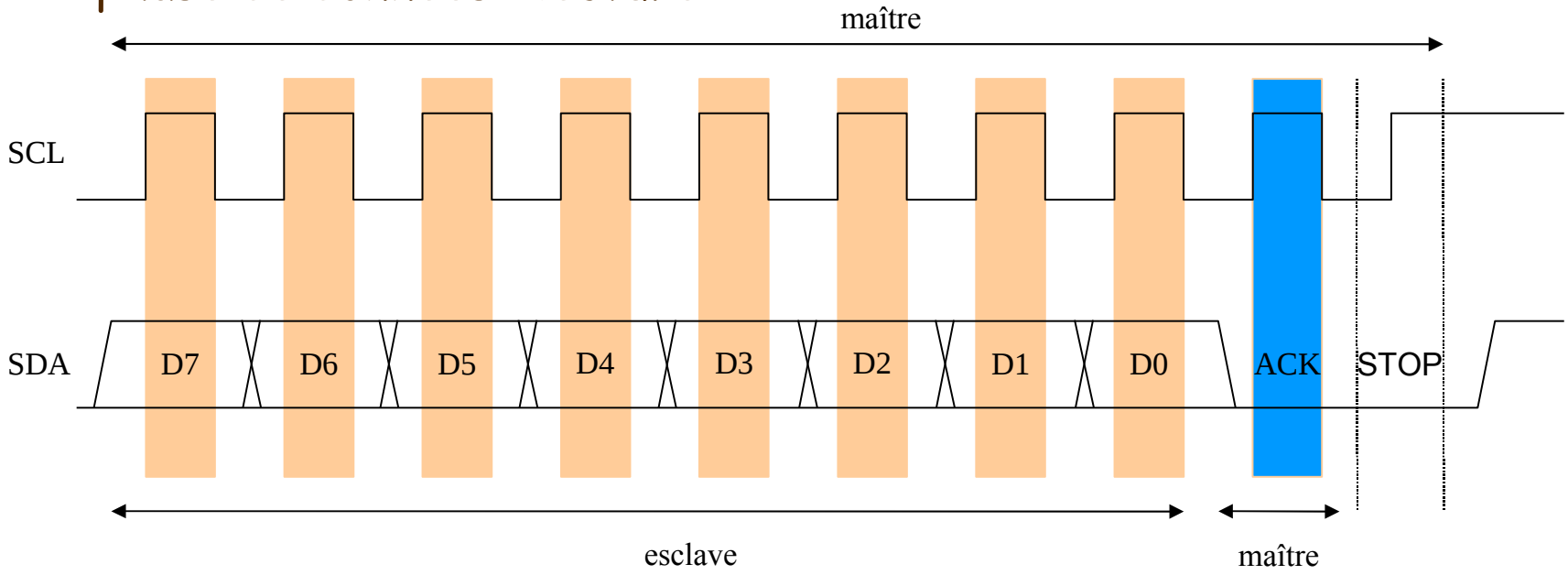
## phase de données : écriture




- RW dans la phase d'adresse = 0
- le maître envoie un ou plusieurs octets à l'esclave qui répond par un ACK
- la transaction se termine par un STOP
- l'esclave interprète les données comme il le veut

# Bus : I2C

## phase de données : lecture




- RW dans la phase d'adresse = 1
- l'esclave envoie un octet
- le maître envoie un ACK pour continuer à lire d'autres octets, un NACK sinon
- la transaction se termine par un STOP

# Bus : I2C

- écritures / lectures combinées
  - un maître peut combiner des écriture et lectures sans passer par l'état STOP
  - exemple :
    - écriture : le maître spécifie à la RAM une adresse à lire
    - lectures : le maître lit les données de la RAM

- un maître peut envoyer un STOP à n'importe quel moment

- utilisation courantes :
  - IO expander
  - flash, eeprom, RAM
  - DDC, SPD
  - DAC et ADC lents
  - contrôleurs vidéo, audio, ...

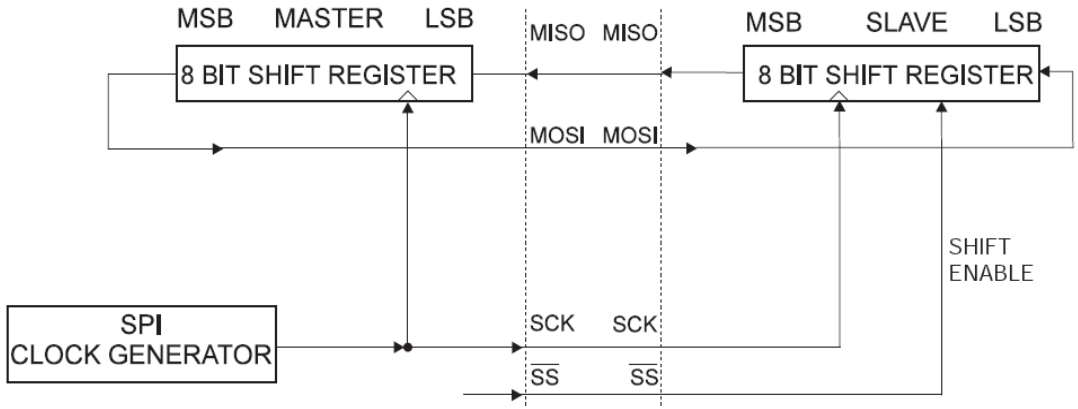
- variante : SMBUS

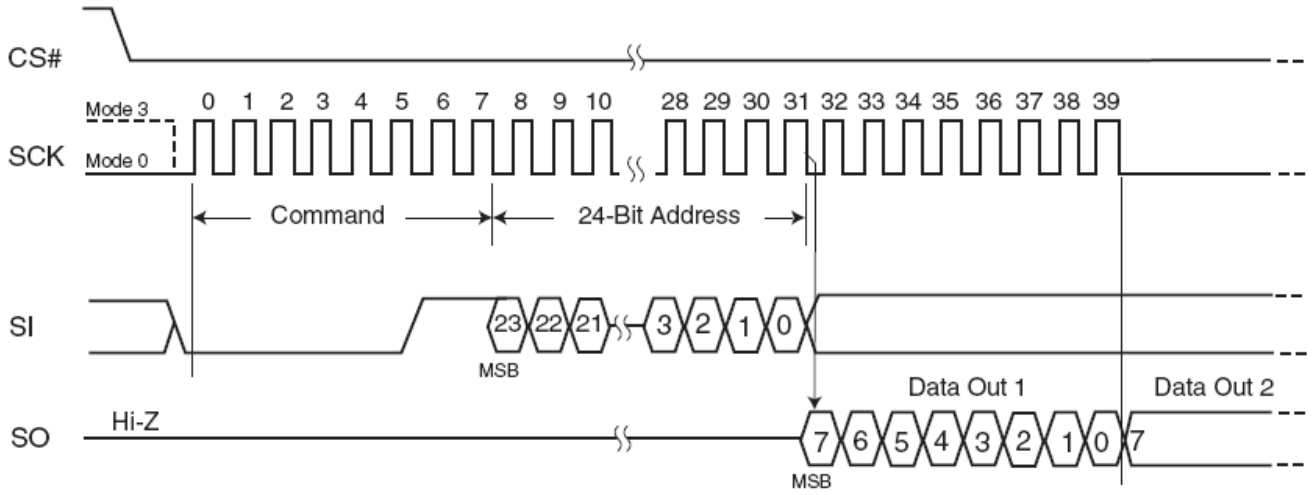
# Bus : SPI

- Serial Péripheral Interface bus, utilisé pour :
  - capteurs : température, pression, ADC, DAC
  - communications : Ethernet, CAN, ...
  - flash et EEPROM
  - écrans LCD
  - cartes MMC et SD
- électriquement
  - 3 ou 4 fils (CS, SCK, MISO, MOSI), point à point ou daisy chain
  - logique CMOS ou TTL
  - 10Mhz en standard
- protocole
  - un maître, un ou plusieurs esclaves
  - full-duplex :
    - MOSI : données transitant du maître vers l'esclave
    - MISO : données transitant de l'esclave vers le maître
    - SCK : horloge
    - CS : chip select

# Bus : SPI

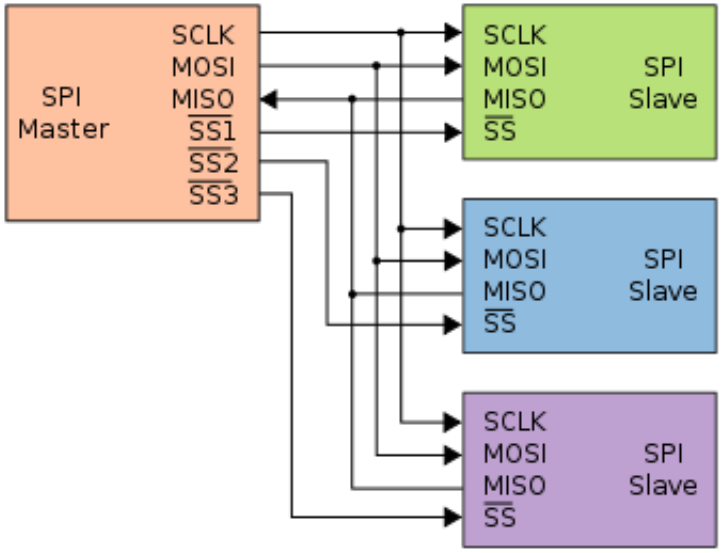
- transmission
  - registres à décalage : échange des données
  - autant de bits que nécessaire (usuellement 8) :
    - codec audio : 16 bits
    - DAC / ADC : 12 bits
    - flash : 8n+32



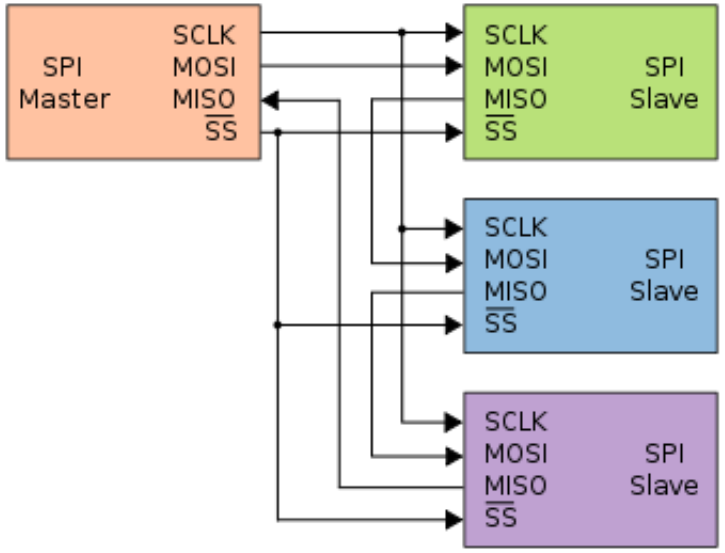



# Bus : SPI

- plusieurs esclaves :
  - point à point
  - daisy chain




wikipedia




wikipedia

# Bus : CAN

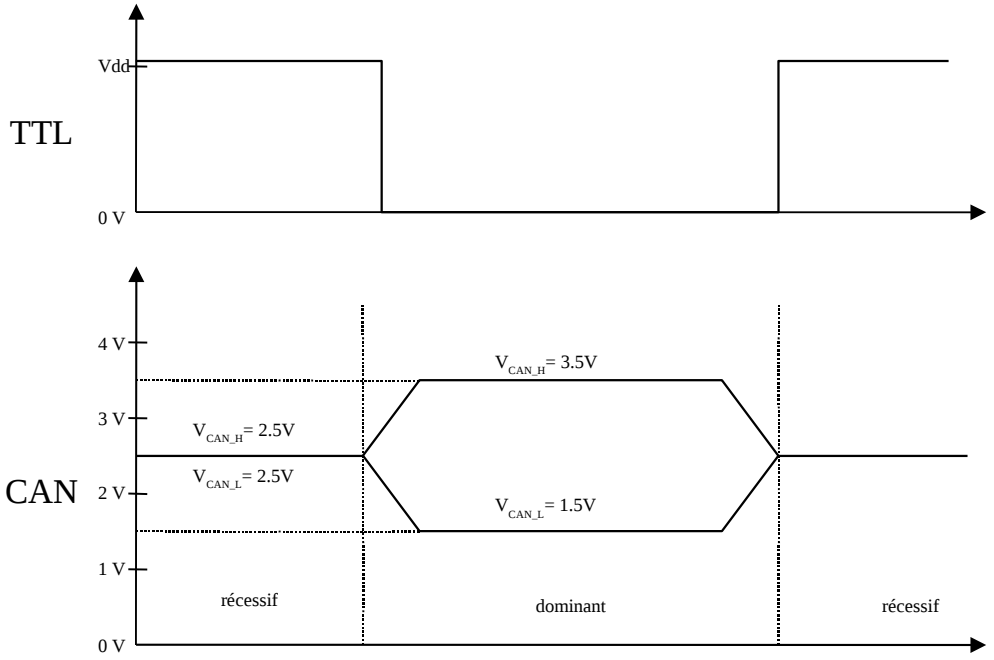
- Controller Area Network
  - très utilisé dans l'automobile et l'industrie
  - robuste, faible latence, temps-réel (CANTT)
  - 2007 : 600 millions de noeuds CAN dans le monde

- électriquement:
  - bus différentiel
  - vitesse variable : 1Mbps (40m) à 10kbps (5km)
  - multidrop

- protocole :
  - multi-maîtres, multi-esclaves
  - arbitration distribuée
  - broadcast : pas d'adressage (content-adressed)
  - messages courts : 8 octets maximum
  - détection et confinement des erreurs

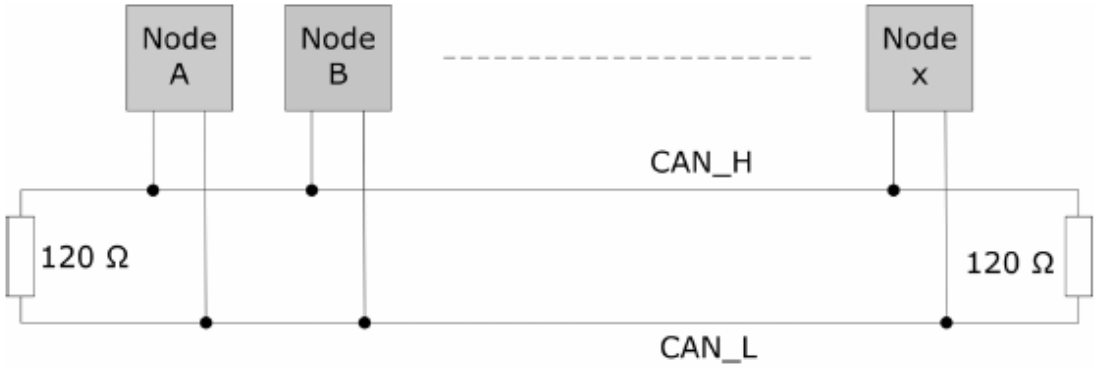
## Signalisation

- deux états
  - 0 - dominant : Vdiff > 2V
  - 1 – récessif, repos : Vdiff = 0V
  - électriquement équivalent à du collecteur ouvert (mais différentiel)
- nécessité d'un transceiver



## Signalisation (suite)

- topologie multidrop
- le bus doit être terminé par des résistances de 120 ohms (cf cours d'intégrité du signal)



# Bus : CAN

## Messages

- 4 types de messages (trames)
  - data frame : « voici le message N et son contenu »
  - request frame : « j'aimerais bien que quelqu'un m'envoie le message N »
    - probablement suivi, quelque temps après, de la data frame contenant la réponse
  - error frame : « il y a eu une erreur »
  - overload frame : « je n'arrive plus à suivre le rythme »

- broadcast, content-adressed
  - les messages sont à destination de tout le monde
  - chaque message possède un indentificateur en plus de son contenu.

    Exemple :
    - message ID 120 : vitesse d'une roue
    - message ID 10 : ordre de freinage d'urgence
  - les cibles ne traitent que les messages dont l'ID les intéresse (filtres)
  - on parle d'adressage « content-based »

# Bus : CAN

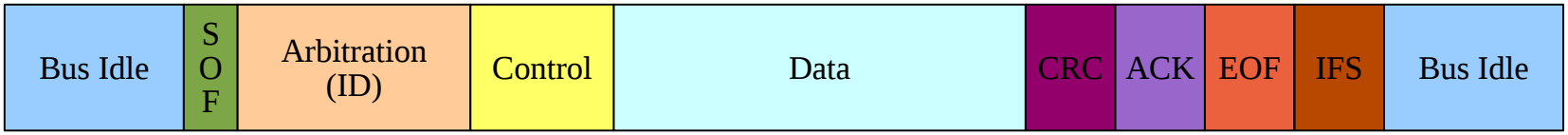
## Arbitration

- quand le bus est libre, n'importe quel maître peut commencer à émettre

- si deux maîtres se mettent à émettre en même temps, celui qui envoie le message de plus petite ID gagne (quizz : pourquoi ?)

- si deux maître émettent des messages de même ID mais de contenu différents, une erreur sera détectée puis reportée.

## Trames de données




- arbitration : 11 ou 29 bits
- control : taille des données utiles
- data : 0 à 8 octets
- CRC : 15 bits
- SOF / EOF : début / fin de la trame
- IFS : espace inter-trames

- bits de stuffing (1 / 5)

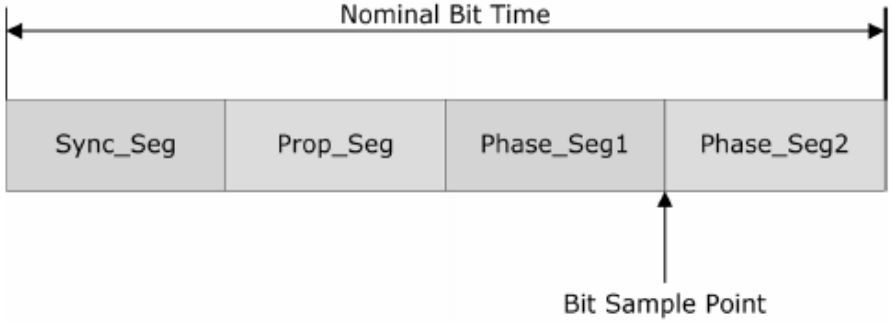
# Bus : CAN

## Gestion des erreurs

- parti-pris : tous les messages sont corrects
- les noeuds confirment la conformité de chaque message (CRC) par l'envoi d'un ACK (dominant)
- 3 possibilités :
  - ACK sans trame d'erreur : tout va bien
  - pas d'ACK : l'émetteur estfautif
  - ACK et trame d'erreur : l'erreur est du côté des récepteurs pas contents
- chaque noeud maintient 2 compteurs d'erreurs : RX et TX
  - une émission / réception correcte : décrémente le compteur (lentement)
  - une émission / réception incorrecte : incrémente le compteur (rapidement)
  - selon la valeur des compteurs, le noeud est actif, passif (réception seule) ou déconnecté du bus
  - d'où détection et confinement des erreurs

## Resynchronisation

- bus asynchrone : dérive des horloges
- une resynchronisation au niveau du bit
  - à chaque SOF
  - à chaque front descendant
- bit divisé en 4 parties, chaque partie en quantum




- en fonction de la position des fronts descendants :
  - PHASE_SEG1 peut être augmenté (de SJW)
  - PHASE_SEG2 peut être diminué (SJW)

## Resynchronisation (suite)

- on programmera les contrôleurs CAN pour que les équations suivantes soient vérifiées (df : tolérance max d'un oscillateur) :

$$\begin{cases} df \le \dfrac{min(PHASESEG1, PHASESEG2)}{2\cdot(13\cdot t_{bit} - PHASESEG2)} \\ df \le \dfrac{SJW}{20\cdot t_{bit}} \\ PROPSEG \ge t_{\text{prop\_A}} + 2\cdot t_{bus} + t_{prop\_B} \end{cases}$$