

# ***Les Bus et Entrées/Sorties***

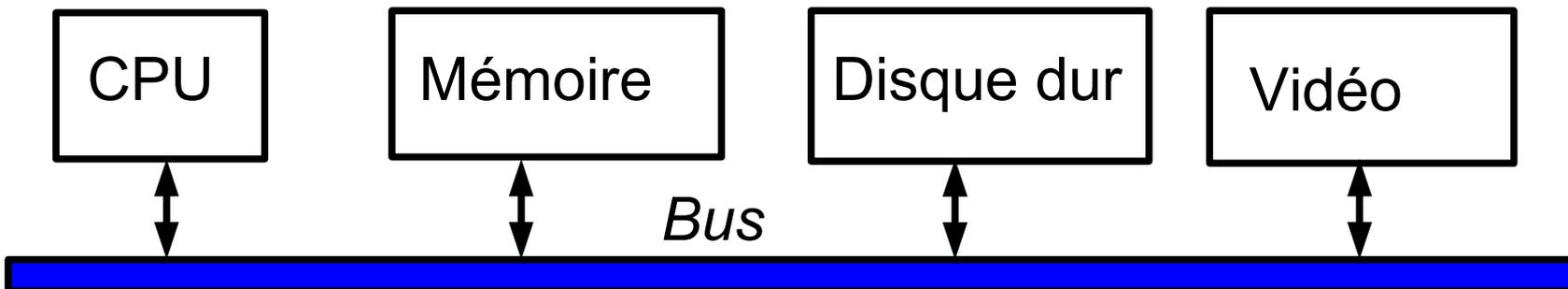
Eric Cariou

*Université de Pau et des Pays de l'Adour  
Département Informatique*

Eric.Cariou@univ-pau.fr

# Bus

- ◆ Les systèmes/éléments dans un ordinateur sont reliés par
  - ◆ Un ensemble de câbles/lignes faisant transiter les informations (signaux électriques : bits)
  - ◆ Ces câbles sont partagés par tous les éléments
- ◆ Bus
  - ◆ Relie plusieurs systèmes via le même câblage électrique



# *Lignes d'un bus*

- ◆ Câbles : 3 ensembles pour un même bus
  - ◆ Lignes d'adresse
    - ◆ Pour identifier l'élément connecté sur le bus auquel on veut accéder
  - ◆ Lignes de données
    - ◆ Pour les données échangées entre les éléments via le bus
  - ◆ Lignes de contrôle
    - ◆ Pour gestion, contrôle du bus
      - ◆ Commande à réaliser : lecture, écriture ...
      - ◆ Signal d'horloge de l'émetteur...
      - ◆ Assurer la validité des données et adresses sur les autres lignes...

# *Maîtres/esclaves*

- ◆ On peut classer les éléments connectés à un bus en 2 catégories
  - ◆ Les esclaves
    - ◆ Ils sont passifs et répondent à des requêtes
      - ◆ Exemple : mémoire
  - ◆ Les maîtres
    - ◆ Ils sont actifs et initient des requêtes
    - ◆ Ils prennent le contrôle du bus
      - ◆ Exemple : processeur
    - ◆ Il ne peut pas y avoir 2 maîtres actifs simultanément
      - ◆ Besoin d'arbitrage

# *Transaction*

- ◆ Transaction pour communication entre maître et esclave
  - ◆ Obtention du contrôle du bus par le maître
  - ◆ Envoi d'une adresse
    - ◆ Qui identifie l'esclave
      - ◆ Exemple : la mémoire centrale
    - ◆ Et identifie des éléments chez l'esclave
      - ◆ Exemple : une adresse en mémoire
  - ◆ Envoi d'une requête (lecture, écriture ...)
  - ◆ Envoi de données vers l'esclave
    - ◆ Ou/et envoi de données par l'esclave
  - ◆ Libération du bus

# *Transfert*

- ◆ Transfert des données
  - ◆ Précision des adresses et des données
    - ◆ Lecture/écriture multiplexée
      - ◆ On envoie d'abord l'adresse
      - ◆ Puis on envoie/reçoit ensuite les données
    - ◆ Lecture/écriture non-multiplexée
      - ◆ Adresses et données sont envoyées en même temps
  - ◆ Envoi/réception des données
    - ◆ Donnée par donnée
    - ◆ Par blocs de données
      - ◆ On précise une adresse
      - ◆ On envoie/reçoit un bloc de données qui sera écrit/lu à partir de cette adresse et aux adresses contiguës

# *Contrôle d'accès au bus*

- ◆ Arbitrage
  - ◆ Pour gérer l'accès au bus si plusieurs maîtres connectés
  - ◆ Définit la politique d'accès des maîtres
    - ◆ Un seul maître contrôle le bus à la fois
- ◆ Propriétés à respecter
  - ◆ Priorité
    - ◆ Certains maîtres sont plus prioritaires que d'autres
  - ◆ Équité
    - ◆ Tous les maîtres auront leur requêtes d'accès satisfaites au bout d'un temps fini

# *Contrôle d'accès au bus*

- ◆ Arbitrage statique
  - ◆ Chaque maître obtient le droit de contrôle du bus à tour de rôle
  - ◆ Avantages
    - ◆ Très simple à mettre en oeuvre
  - ◆ Inconvénients
    - ◆ Passe la main aux maîtres ne désirant pas accéder au bus
    - ◆ Pas de gestion des priorités

# *Contrôle d'accès au bus*

- ◆ Arbitrage dynamique
  - ◆ Quand un maître veut accéder au bus, il en fait la requête
  - ◆ La gestion des accès est alors faite dynamiquement
    - ◆ Les maîtres font leur demande d'accès en envoyant un signal de contrôle particulier
  - ◆ Plusieurs variantes pour gérer des demandes simultanées d'accès
    - ◆ Centralisé
    - ◆ Par chaînage
    - ◆ Décentralisé

# *Contrôle d'accès au Bus*

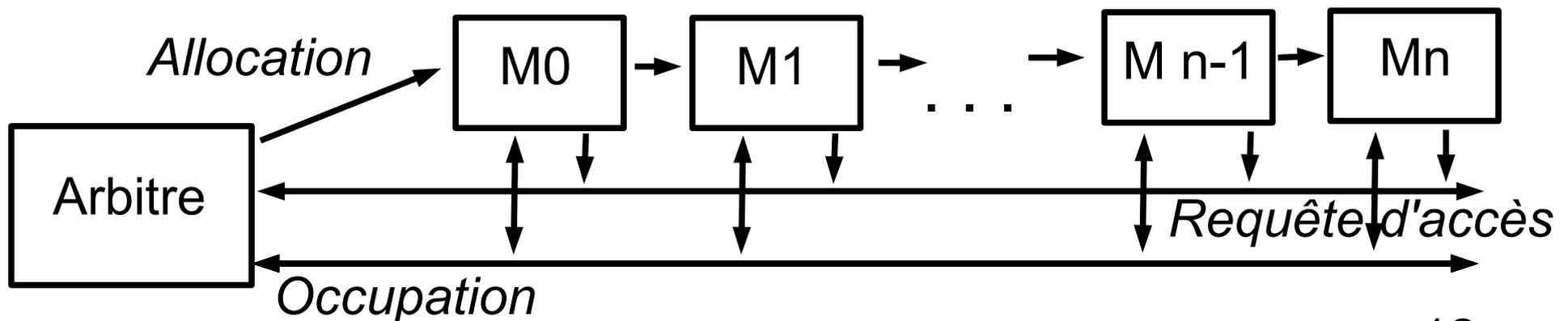
- ◆ Arbitrage centralisé
  - ◆ Un élément particulier centralise toutes les requêtes d'accès au bus des maîtres
    - ◆ Lignes spécialisées sur le bus pour ces requêtes
  - ◆ Il connaît les priorités de chacun
  - ◆ A partir de là, c'est lui qui donne l'accès au bus au maître le plus prioritaire
    - ◆ Lignes spécialisées sur le bus pour cette attribution

# *Contrôle d'accès au Bus*

- ◆ Arbitrage par chaînage
  - ◆ Les maîtres forment une chaîne avec les plus prioritaires au début de la chaîne
  - ◆ Il y a une ligne de requête et une ligne d'occupation du bus
  - ◆ Un élément voulant accéder au bus envoie un signal sur la ligne de requête
  - ◆ L'arbitre envoie alors un jeton d'allocation le long de la chaîne
  - ◆ Le premier maître qui veut accéder au bus récupère l'allocation et utilise le bus
  - ◆ Il envoie un signal sur la ligne d'occupation pour préciser que le bus est utilisé

# Contrôle d'accès au Bus

- ◆ Arbitrage par chaînage (suite)
  - ◆ Quand le maître actif à fini son opération, il désactive la ligne d'occupation
  - ◆ Si la ligne de requête est encore active, le maître renvoie un autre jeton d'allocation
  - ◆ Pour éviter la famine (un maître n'a jamais accès au bus)
    - ◆ On interdit à un maître venant d'accéder au bus de faire une nouvelle requête d'accès tant que la ligne de requête n'est pas inactive



# *Contrôle d'accès au Bus*

- ◆ Arbitrage décentralisé
  - ◆ Plusieurs lignes/bit de priorité sont utilisées
    - ◆ 4 lignes : 16 niveaux de priorité
  - ◆ Si un élément veut contrôler le bus
    - ◆ Compare son niveau de requête à celui des requêtes en cours
    - ◆ Pourra déterminer s'il pourra contrôler le bus

# *Accès synchrone/asynchrone*

- ◆ Émission d'informations sur les différentes lignes en 2 modes
  - ◆ Synchrone : signal d'horloge
    - ◆ Horloge partagée entre l'émetteur et le récepteur
      - ◆ Signal d'horloge généralement émis sur une ligne
      - ◆ On sait le type d'information qui passe à tel cycle d'horloge dans une transaction (donnée, adresse, contrôle)
  - ◆ Asynchrone : pas de signal d'horloge
    - ◆ Nécessité d'envoyer des informations de contrôle supplémentaires
      - ◆ Prévenir du début d'une émission
      - ◆ De la bonne réception de la part du récepteur (acquiescement)
    - ◆ But : créer une synchronisation entre maître et esclave

# *Accès synchrone/asynchrone*

## ◆ Synchrone

- ◆ A réserver pour des petites distances : < 50 cm
  - ◆ Car problème de dispersion de signal d'horloge et de synchronisation sur longue distance
  - ◆ Utilisé généralement
    - ◆ Pour bus internes au CPU
    - ◆ Bus rapides (mais court)

## ◆ Asynchrone

- ◆ Permet des bus plus longs
  - ◆ Plus lents que le synchrone
- ◆ Permet de connecter des éléments fonctionnant à des vitesses différentes

# Codage des données

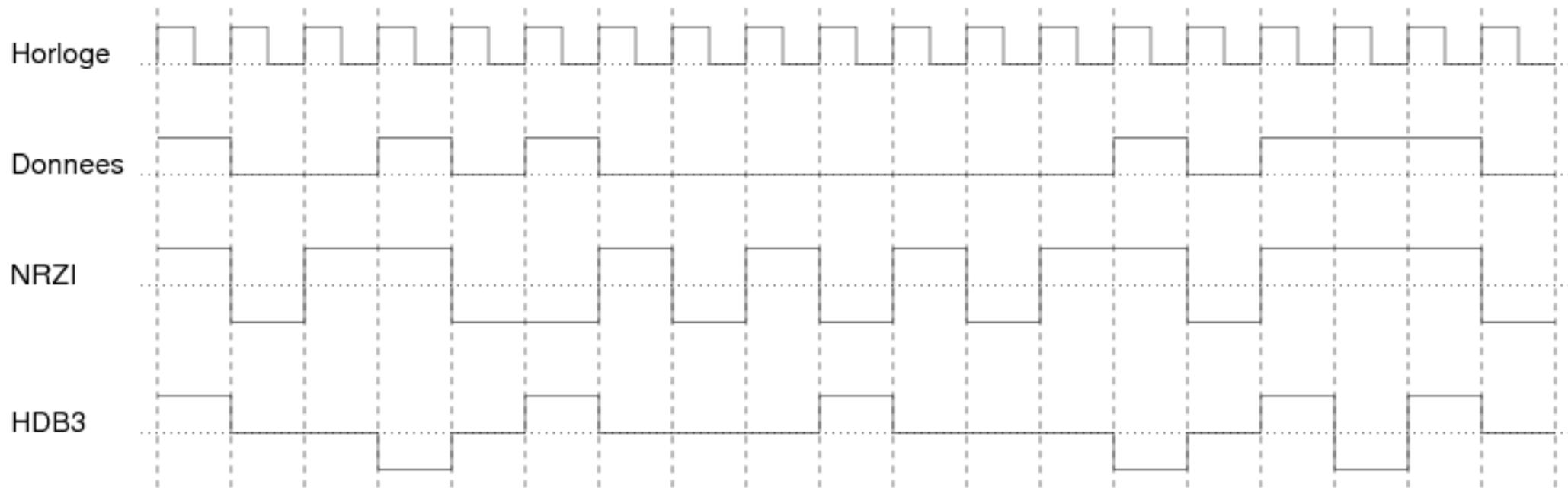
- ◆ Dans un composant électronique, le codage physique des données binaires se fait simplement
  - ◆ 0 est codé par l'absence de courant pendant un cycle d'horloge
  - ◆ 1 est codé par un voltage positif (+V) pendant un cycle d'horloge
- ◆ Ce codage n'est pas adapté pour toutes les transmissions
  - ◆ Sur une plus longue distance, pour des raisons électriques, la valeur moyenne du signal doit être nulle
    - ◆ On doit envoyer des -V en nombre équivalent aux +V
  - ◆ En asynchrone, le récepteur doit retrouver dans le signal de données la fréquence d'horloge d'émission et se synchroniser dessus pour lire correctement les données
    - ◆ Doit voir où commence un bit et trouver combien de temps dure un cycle d'horloge
    - ◆ N'est possible que si le signal varie régulièrement
- ◆ Plusieurs codages existent pour gérer ces problèmes

# Codage des données

- ◆ USB utilise un codage NRZI (Non Return to Zero Inverted)
  - ◆ Un bit envoyé est codé par +V ou -V, pas de courant nul
  - ◆ Un 0 inverse le voltage utilisé pour le bit précédent
    - ◆ Si +V, on envoie -V, si -V, on envoie +V
  - ◆ Un 1 est codé avec le même voltage que le bit précédent
  - ◆ Inconvénient : si une longue suite de 1, le signal ne varie pas et cela pose problème pour rester synchronisé en réception
    - ◆ Pour gérer cela, un 0 (une inversion) non compris dans les données est envoyé au bout de six 1 consécutifs
    - ◆ Envoi supplémentaire de données non utiles
- ◆ HDB3 est utilisé dans les réseaux télécoms en Europe
  - ◆ Un 0 est codé par l'absence de courant et un 1 par un voltage inversé du 1 précédent
  - ◆ S'il y a quatre 0 de suite, le quatrième est envoyé avec le voltage du 1 précédent pour faire varier le signal et ne pas le confondre avec un 0

# Codage des données

- ◆ Exemple de codage de données pour NRZI et HDB3



- ◆ Il existe de nombreux autres codages :  
AMI, Manchester, ...

# *Performances*

- ◆ Les performances du bus sont caractérisées par
  - ◆ Le largeur du bus
    - ◆ Nombre de bits/lignes du bus
  - ◆ La fréquence de fonctionnement du bus
    - ◆ Nombre d'émission/reception de bits par ligne par seconde
  - ◆ Type de transferts
    - ◆ Par bloc plus efficace pour grosses données
  - ◆ Synchronicité du bus
  - ◆ Nombre d'éléments que l'on peut connecter sur le bus
    - ◆ Partage de la bande passante ou attente de disponibilité du bus

## *Exemple : bus PCI*

- ◆ PCI (Peripheral Component Interconnect)
  - ◆ Pour connexion de carte réseau, son, contrôleurs de disques ...
  - ◆ Fréquence : 33 Mhz ou 66 Mhz
  - ◆ Largeur : 32 ou 64 bits
    - ◆ Débit max théorique
      - ◆ 127 Mo/s en 33 Mhz et 32 bits
      - ◆ 509 Mo/s en 66 Mhz et 64 bits
  - ◆ Multiplexage adresses et données
    - ◆ Pas de lignes séparées pour données et adresses
  - ◆ Envoi synchrone par blocs de données
  - ◆ Arbitrage centralisé

# *Exemple : bus USB*

- ◆ USB (Universal Serial Bus)
  - ◆ Connexion de périphériques extérieurs
    - ◆ Imprimantes, scanners, APN, disque dur ...
    - ◆ Connexion et déconnexion à chaud
  - ◆ Série
    - ◆ Une seule ligne pour émission des informations
  - ◆ Débit max, selon les versions
    - ◆ 1,5 Mo/s en version 1.1, 60 Mo/s en 2.0, 600 Mo/s en 3.0 et 1,2 Go/s en 3.1
  - ◆ 127 périphériques simultanés au plus
    - ◆ Bande passante partagée entre tous
  - ◆ Longueur max des câbles : 5 mètres

# *Entrées/Sorties*

- ◆ Dispositif d'entrées/sorties
  - ◆ Clavier, souris, disque dur, ...
  - ◆ Un contrôleur d'entrées/sorties gère tout cela
- ◆ Gestion des entrées/sorties, 2 modes
  - ◆ Interrogatif (polling)
    - ◆ Le CPU interroge le contrôleur à intervalles réguliers pour connaître l'état d'une E/S
  - ◆ Interruptif
    - ◆ Le CPU est informé par le contrôleur qu'une tâche est à réaliser via l'envoi d'une requête d'interruption
    - ◆ Le CPU s'arrête et active le contrôleur qui réalise la tâche d'E/S

# *Entrées/Sorties*

- ◆ Choix entre les 2 modes
  - ◆ Dépend de la nature du périphérique d'entrée/sortie
- ◆ Exemples
  - ◆ Souris
    - ◆ En polling, 60 fois par seconde (60 Hz)
    - ◆ Car même si la récupération de l'état de la souris prend plusieurs dizaines ou centaines de cycles d'horloge
      - ◆  $60 \times \text{nbCycles}$  est négligeable devant quelques millions (les CPU ont des fréquences de plusieurs Ghz)
  - ◆ Disque dur
    - ◆ Avec interruption
      - ◆ Car débit très important, prend trop de temps de regarder si une opération de lecture est en cours ou pas