heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Bus de terrain

Aperçu de Can et CanOpen

heig-vd

Thèmes abordés

- Aperçu rapide de
  - Can et CanOpen
- Souligner leurs caractéristiques originales

heig-vd

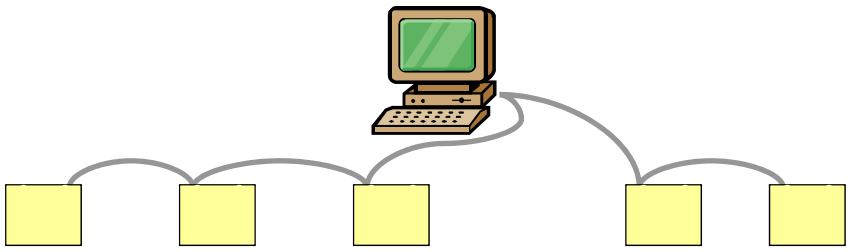
CanOpen
Présentation

- Nom
  - CAN : Controller Area Network.
  - CanOpen : Couche 7 (application) au dessus de CAN.
- Origine
  - CAN a été développé pour l'automobile en 1986.
  - Idée d'un réseau interne aux véhicules pour simplifier le câblage.
- Standardisation
  - ISO 11898
  - CanOpen : EN 50325-4
- Organisation
  - Can In Automation : www.can-cia.org

heig-vd

Can - CanOpen
Utilisation typique

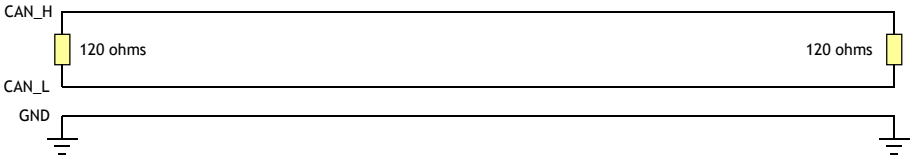
- Topologie BUS



heig-vd

Can
Couche Physique

- Câble
  - 1 paire torsadée + 1 masse
    - GND, CAN_L, CAN_H
  - Blindage recommandé
    - Pour des longues distances
    - Pour des environnements bruyants
- Résistances de terminaison
  - Simples résistances de terminaison de 120 ohms à chaque extrémité.

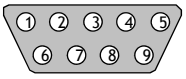


heig-vd

Can Open
Couche Physique

- Connecteur

Pin #	Signal Names	Signal Description
1	Reserved	Upgrade Path
2	CAN_L	Dominant Low
3	CAN_GND	Ground
4	Reserved	Upgrade Path
5	CAN_SHLD	Shield, Optional
6	GND	Ground, Optional
7	CAN_H	Dominant High
8	Reserved	Upgrade Path
9	CAN_V+	Power, Optional

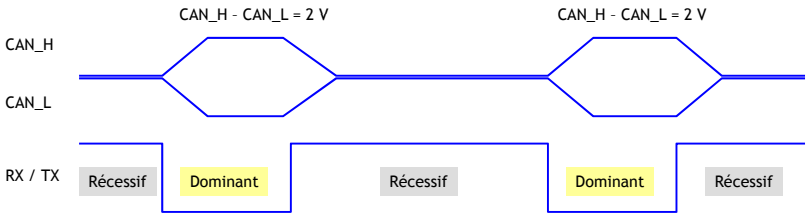


heig-vd

Can

Couche Physique – Modulation du signal

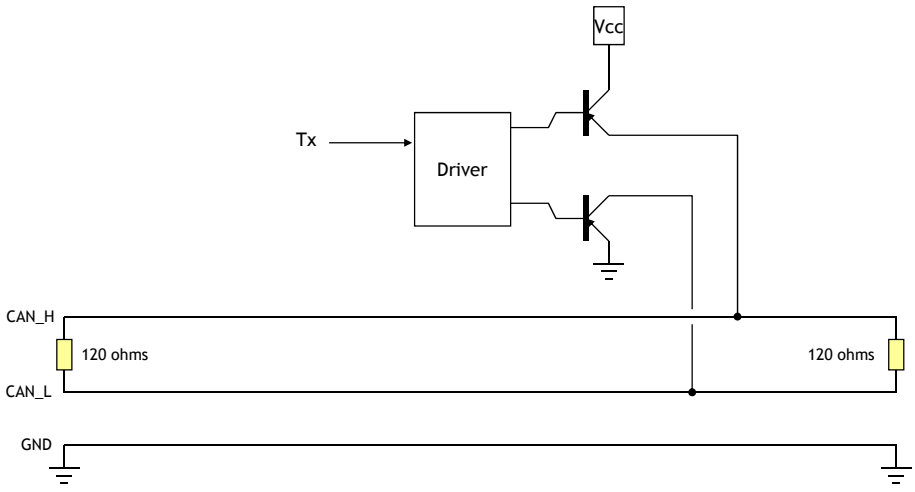
- Traduction électrique des états
  - Etat 0 : Imposition d'une différence de potentiel entre CAN_H et CAN_L.
  - Etat 1 : Sortie laissée en haute impédance.
- Lorsque plusieurs stations émettent simultanément un 0 et un 1
  - L'état résultant de la ligne est 0.
  - On dit que l'état 0 est dominant, l'état 1 est récessif.
- Traduction électrique des états



heig-vd

Can

Couche Physique – Modulation du signal – Transmission différentielle



heig-vd

Can

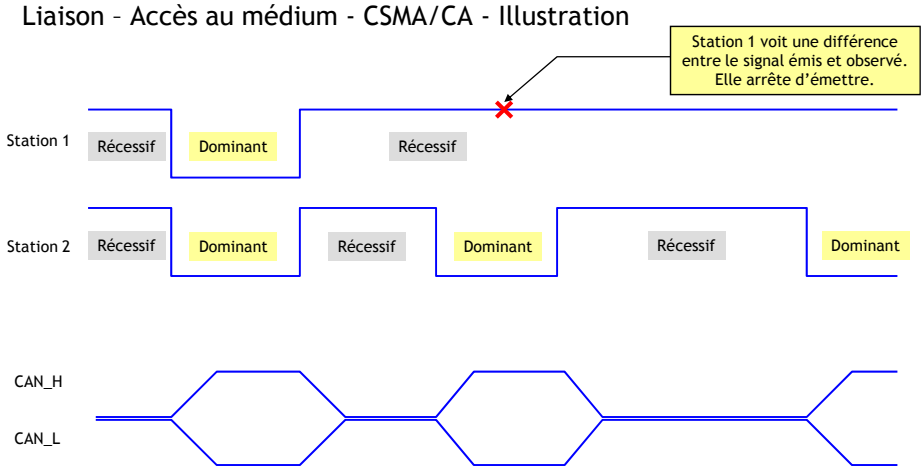
Liaison – Accès au médium - CSMA/CA

- CSMA/CA
  - Carrier Sense Multiple Access with Collision Avoidance .
  - Détection de porteuse avec évitement de collision.
- Principe
  - Avant d'émettre, une station « écoute » et vérifie que le médium est disponible.
  - Si c'est le cas, elle commence à émettre.
  - Pendant l'émission, la station compare ce qu'elle envoie avec ce qu'elle observe sur le câble.
    - En cas de différence, elle arrête immédiatement d'émettre.
- Si 2 stations commencent à émettre simultanément
  - Tant qu'elles émettent la même chose, pas de problème.
  - Dès qu'il y a une différence sur un bit, celle qui envoie le bit dominant peut poursuivre, l'autre s'arrête.

heig-vd

Can

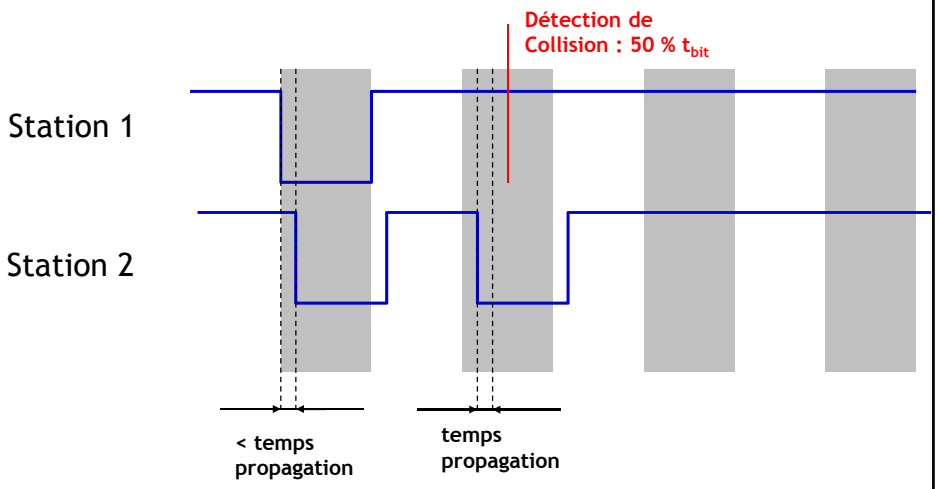
Liaison – Accès au médium - CSMA/CA - Illustration



heig-vd

Can

Conséquence 1 – Lien entre débit et distance



heig-vd

Can

Conséquence 1 – Lien entre débit et distance

- la durée de chaque bit doit être assez longue pour que chaque station ait le temps de détecter la collision
- le signal doit donc pouvoir faire un aller-retour avant ~40% de la durée du bit
  - exemple : à 500 kbits/s, la distance totale est limitée à

$$L_{max} < 200'000'000 / 500'000 \times 0.4 / 2 = 80 \text{ m}$$

Débit	Longueur
1 Mbit/s	40 m
500 Kbit/s	80 m
100 Kbit/s	400 m
20 Kbit/s	1000 m

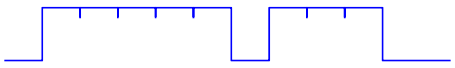
Can

Conséquence 2 – Différentes possibilités pour les échanges de données

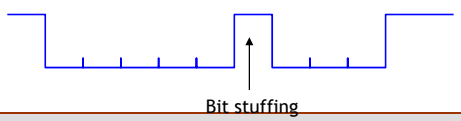
- Mode polling
  - CAN permet de réaliser une interrogation des stations par polling comme Profibus
- Mode évènementiel
  - Une station peut aussi émettre spontanément un message seulement lorsque c'est utile.
  - Meilleure exploitation de la bande passante.
  - Temps de réponse sur évènement plus court.
  - Possibilité d'envoyer un message de synchronisation à tous.
- Mode multi maître naturel
  - Plusieurs maîtres peuvent accéder aux stations
  - Sans moyen de synchronisation supplémentaire

Can
Bit stuffing

- Problème avec les longues séries de bits identiques
  - Le récepteur n'a plus de flancs pour synchroniser l'horloge
- Can utilise le bit stuffing
  - Utilisé si 5 bits consécutifs de même valeur doivent être transmis.
  - L'émetteur introduit automatiquement un bit de valeur opposée.
  - Le récepteur l'élimine automatiquement
  - Exemple : transmission de 8 bits à 1



  - Transmission de 8 bits à 0

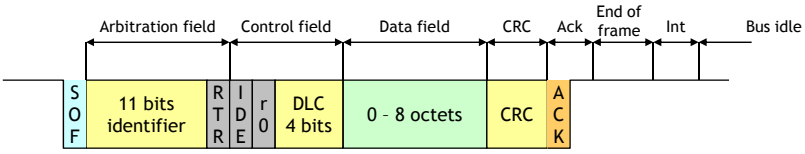


heig-vd

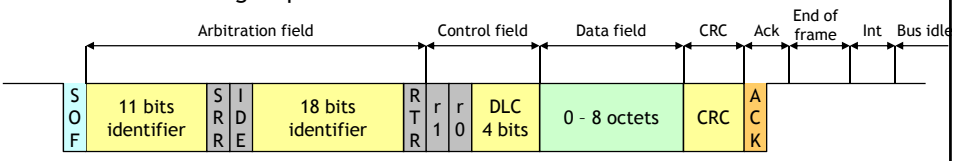
Can

Liaison – Format des messages

Format des messages spécification CAN 2.0A



Format des messages spécification CAN 2.0B



heig-vd

Can

Liaison – Format des messages – Can 2.0 A

Champ	Taille (bits)	Rôle
Start-of-frame	1	Denotes the start of frame transmission
Identifier	11	A (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)
Identifier extension bit (IDE)	1	Must be dominant (0)
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0)
Data length code (DLC)	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

heig-vd

Can

Liaison – Format des messages – Can 2.0 B

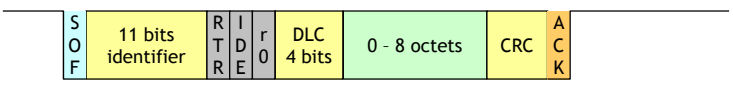
Champ	Taille	Rôle
Start-of-frame	1	Denotes the start of frame transmission
Identifier A	11	First part of the (unique) identifier for the data
Substitute remote request (SRR)	1	Must be recessive (1)
Identifier extension bit (IDE)	1	Must be recessive (1)
Identifier B	18	Second part of the (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)
Reserved bits (r0, r1)	2	Reserved bits (it must be set dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1), any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

heig-vd

Can

Liaison – Détection et gestion des erreurs

- Le CRC des messages qui passent est contrôlé par toutes les stations.
- Si le message est valide
  - Les stations génèrent l'acquittement. Permet de vérifier que le message a été reçu.
    - Ack : valeur dominante
    - Ack delimiter : valeur récessive.
- Si le message est invalide
  - La ou les stations détruisent le message pour toutes les stations.
    - En mettant Ack delimiter à la valeur dominante.
  - L'émetteur attend l'intervalle inter trame, puis réémet.



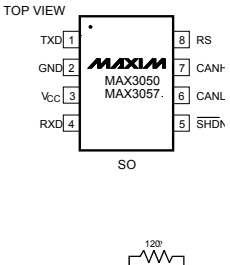
Can

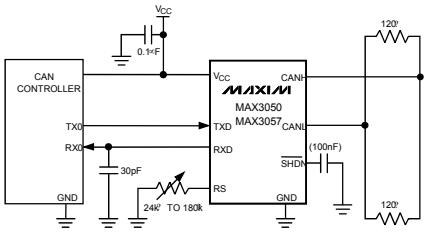
Liaison – Adressage

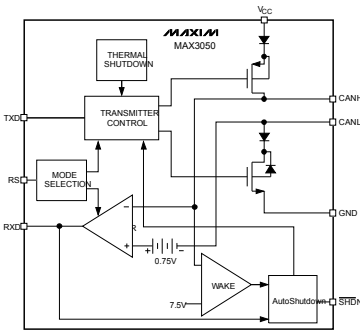
- Principe
  - CAN ne comporte pas de notion d'adresse
  - On utilise en général certains bits du champs d'arbitrage comme bits d'adresse.
  - La plupart des contrôleurs CAN savent « filtrer » les messages reçus sur la base du champ d'arbitration.
- Conséquences
  - Les adresse plus basses ont une priorité plus haute.

Can

Intégration dans une électronique – Drivers de ligne





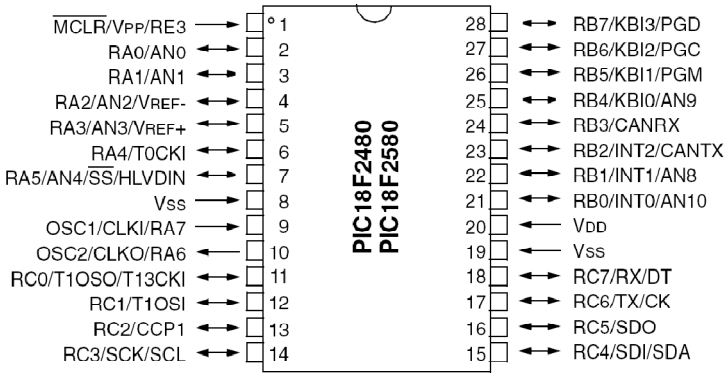


heig-vd

Can

Intégration dans une électronique - Contrôleurs CAN

- Directement intégré dans de nombreux micro contrôleurs
  - Même à très bas coût
  - Exemple : Microchip 8 bits avec interface Can, ~ 5$



heig-vd

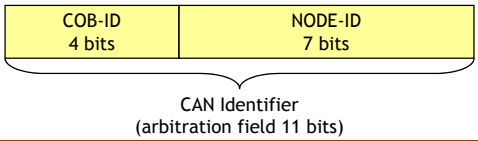
Can

Intégration dans une électronique - Programmation

- Programmation du contrôleur CAN à travers une série de registres internes.
  - Registres de configuration.
  - Registres d'envoi et de réception de données.
- Génération automatique d'une interruption à la réception d'un message.
- La complexité reste en général raisonnable.

CanOpen
Un protocole de la couche application

- CanOpen
  - Utilise le format Can 2.0 A.
  - Définit la structure détaillée des messages.
  - Protocole applicatif largement paramétrable.
  - Donc assez complexe.
  - Permet d'utiliser efficacement CAN selon les besoins applicatifs.
- L'identificateur CAN est divisé en 2 parties
  - COB-ID : Communication Object ID, sur 4 bits
  - Node-ID : sur 7 bits. Adresse du nœud, 0 à 127.



CanOpen
Quelques COB-ID

COB-ID	Usage	Construction
000h	NMT (Network Management)	
001h	Global Failsafe Command	
080h	SYNC	
081h-0FFh	Emergency	80h + NodeID
181h-1FFh	Transmit PDO1	180h + NodeID
201h-27Fh	Receive PDO1	200h + NodeID
…	…	…
581h-5FFh	Transmit SDO	580h + NodeID
601h-67Fh	Receive SDO	600h + NodeID

Priorité la plus haute

Priorité la plus basse

heig-vd

CanOpen
SDO – Service Data Object

- Télégrammes permettant d'envoyer ou de lire des informations de configuration.
  - Utilisés pendant les phases d'initialisation.
- Permettent notamment à une application de configurer
  - Quelles données de processus doivent être transmis (PDO).
  - Sous quelle conditions ces données doivent être envoyées.
- Ces informations sont écrites dans l'Object Dictionnary
  - Chaque esclave comporte un Object Dictionnary.
  - Cet object Dictionnary configure le comportement de l'esclave.

heig-vd

CanOpen
PDO – Process Data Object

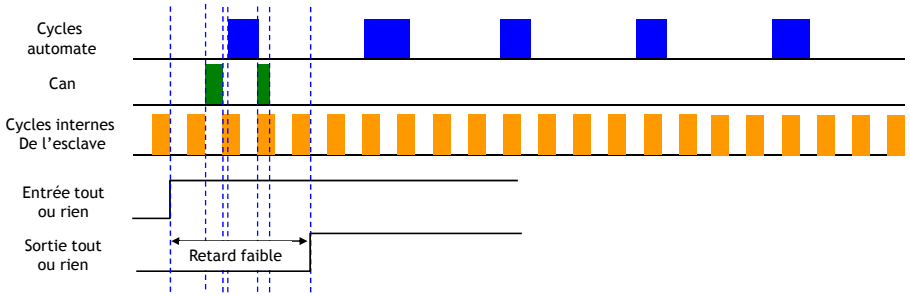
- Un PDO véhicule l'information du processus
  - Etat des entrées analogiques et tout ou rien.
  - Etat des sorties.
- Selon la configuration, un PDO est envoyé
  - Sur réception d'un télégramme d'une autre station.
  - Sur un évènement interne de l'esclave :
    - changement d'état d'une entrée.
  - Sur réception du message SYNC.
  - De façon périodique après N SYNC.
  - Intervalle minimum entre 2 envois successifs, pour éviter le surcharge.
- Avantage
  - Un seul message SYNC peut déclencher l'envoi de tous les PDO des différentes stations.
  - Gestion efficace de la bande passante.

heig-vd

Can – CanOpen

Avantages du modèle évènementiel

- Au niveau de la charge du bus
  - Un message n'est envoyé que lors d'un changement d'état
- Au niveau de la rapidité
  - Si le bus n'est pas chargé, notification immédiate.
  - Pas de retard supplémentaire dû à un temps de cycle.

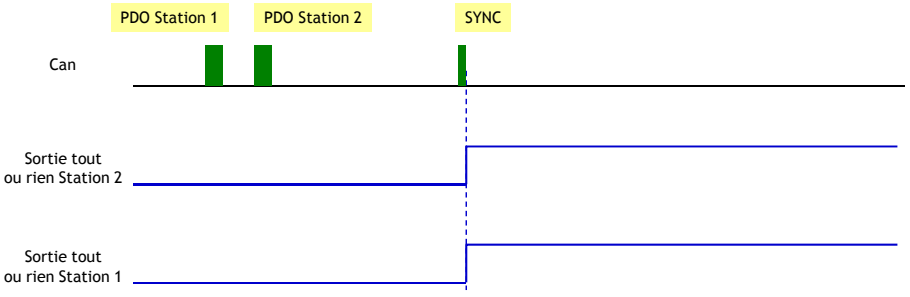


heig-vd

Can – CanOpen

Capacités de synchronisation temporelle

- Synchronisation par un message en diffusion
  - Tous les esclaves le reçoivent presque simultanément
  - Permet de coordonner le démarrage d'un mouvement.
    - Problème en cas de réémission du message
  - Ce message devrait être le plus prioritaire.



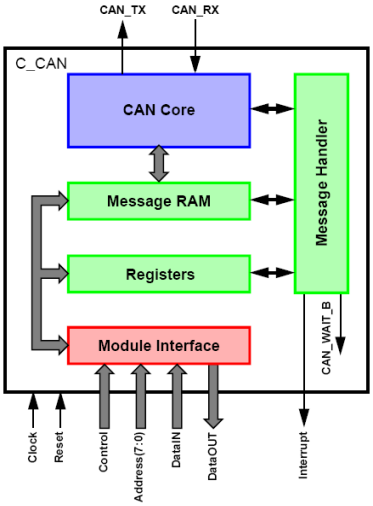
Can – CanOpen

Faiblesse du modèle évènementiel

- Non déterminisme
  - Le temps de propagation d'un évènement dépend de la charge du bus
  - Si le bus est très chargé, un message non prioritaire peut prendre un temps très long à arriver.
  - Problème observable dans la pratique.
- Parades
  - Eviter de répéter trop souvent les PDO
    - Imposer un temps minimum entre 2 émissions d'un même PDO
  - Ne garantit rien sur un bus avec de nombreuses stations.

Implémentation du bus CAN

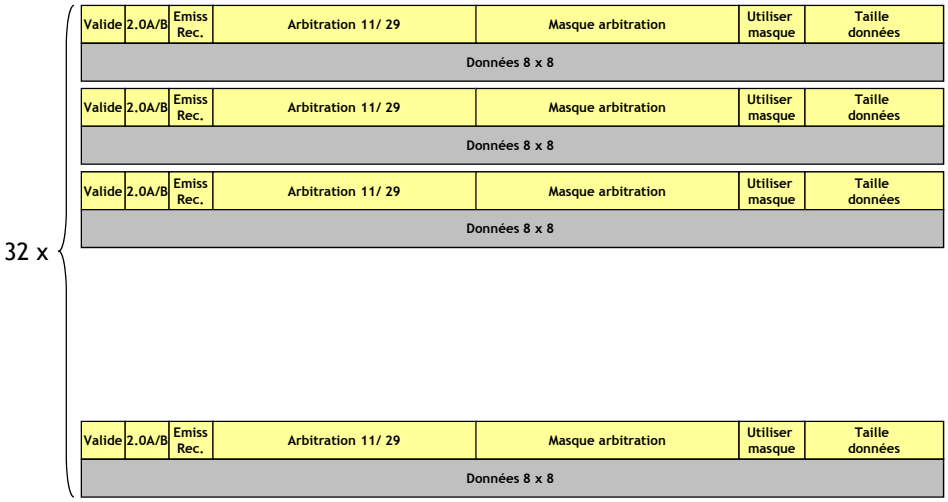
Exemple : Programmation du contrôleur Bosch



heig-vd

Implémentation du bus CAN

Exemple : Programmation du contrôleur Bosch – Message RAM



heig-vd

Implémentation du bus CAN

Exemple : Programmation du contrôleur Bosch

- Message RAM : 32 objets message
  - Zones mémoire pour configurer un message.
  - Options :
    - Objet message utilisé ou non
    - Can 2.0A ou Can 2.0B
    - Emission ou réception
    - Champ d'arbitration (11 ou 29 bits)
    - Masque d'arbitration (réception, 11 ou 29 bits)
    - Utiliser ou non le masque
    - Taille de données

heig-vd

Implémentation du bus CAN

Exemple : Programmation du contrôleur Bosch

- Procédure à suivre pour l'initialisation
  - Configurer les objets message utilisés.
  - Activer le contrôleur CAN
- Envoi de message
  - Ecrire les données dans l'objet message correspondant.
  - Activer la demande d'envoi de cet objet message
- Réception
  - Lire le bit « Message arrivé »
  - Si vrai, lire les données

heig-vd

Implémentation du bus CAN

Exemple : Programmation du contrôleur Bosch

- Bibliothèque fournie en laboratoire
  - Simplifie la mise en œuvre du contrôleur CAN Bosch sur Silabs
- Initialisation

```
void can_configurer_objet_message(
    char numero_objet_message, // 1..32
    char message_valide, // 0 ou 1
    char can_2_0b, // 0 (CAN 2.0A) ou 1 (CAN 2.0B)
    char emission,  // 0 ou 1
    long champ_arbitration, // 11 ou 29 bits utilisés selon type_can
    long masque_arbitration, // 11 ou 29 bits utilisés selon type_can
    char utiliser_masque, // 0 ou 1
    char taille_donnees // taille des donnees du message en octet
);

void can_activer_controleur(
    char autoriser_interruption // 0 ou 1
);
```

heig-vd

Implémentation du bus CAN

Exemple : Programmation du contrôleur Bosch

- Envoi

```
void can_envoyer_message(
    char numero_objet_message, // 1..32
    char * donnees,
    char taille_donnees
);

char can_message_envoye(
    char numero_objet_message // 1..32
); // retourne 0 / 1
```


heig-vd

Implémentation du bus CAN

Exemple : Programmation du contrôleur Bosch

- Réception

```
char can_message_recu(
    char numero_objet_message // 1..32
); // retourne 0 / 1

char can_lire_message(
    char numero_objet_message,  // 1..32
    char * donnees,
    char taille_tampon_donnees
); // retourne la taille lue
```

heig-vd

Implémentation du bus CAN
Exemple : Programmation du contrôleur Bosch

- Gestion des erreurs

```
char can_erreur(
); // retourne 0 / 1


void can_effacer_erreur(
);
```


heig-vd

Can
Analyse

- Can
  - Performance du principe évènementiel.
  - Non déterminisme, sauf pour le message de priorité maximale.
  - Simplicité d'intégration dans une électronique.
  - Très économique.
- CanOpen
  - Protocole assez répandu.
  - Tire bien parti de Can. Très adaptable aux besoins de l'application.
  - Une certaine complication au niveau logiciel.
  - Il existe des bibliothèques toute prêtes en C.
- Alternatives
  - Développement d'un protocole propriétaire sur CAN.
  - Très efficace lorsque l'interopérabilité n'est pas nécessaire
    - Voie suivie par de nombreuses sociétés.

heig-vd

Qu'avons-nous appris ?

- CAN
  - Couche 1 et 2 du modèle OSI.
  - Un protocole simple et efficace.
  - Facile à exploiter à moindre coût dans toute carte à microprocesseur.
- CanOpen
  - Couche 7 du modèle OSI
  - Une solution un peu compliquée, mais très configurable.
- Can Propriétaire
  - Solution à considérer dans les développements électroniques.
  - Permet de tirer parti des avantages de CAN.
  - Sans la complexité de CanOpen.

heig-vd

Vos questions


Exercices
CAN