C1 TP2: Utlisation du protocole CAN

Felipe Morán Juan Sánchez

11/11/2018

1 Masque d'acceptation et code d'acceptation

Pour garantir la bonne réception des trames CAN il est nécessaire calculer la masque d'acceptation et le code d'acceptation pour filtrer correctement les messages utiles et ignorer celles dont on n'en a pas besoin. Il est toujours possible de faire cette filtrage en logiciel, pourtant Il est souvent envisageable d'utiliser le support matériel quand disponible pour réduire la complexité logicielle et augmenter la performance.

La masque et code d'acceptation de la filtrage matérielle dont on dispose ont la taille de 8 bits, mais l'adresse a une taille de 11 bits, donc pour cette raison on ne considère que les 8 bits de poids plus fort de l'adresse pendent la filtrage.

Le calcul de la masque d'acceptation consiste simplement en réaliser l'opération AND bit à bit entre tous les adresses qu'on veut recevoir, en suite l'opération OR bit à bit entre tous les adresses et finalement l'opération XOR entre les deux résultat précédant. Le calcul du code d'acceptation consiste en réaliser l'opération AND bit à bit entre l'inverse bit à bit de la masque d'acceptation et un adresse quelconque de la liste des adresses acceptés.

Ci dessous quelques exemples:

1.1 Accepter adresse: 0x310

Comme décrit ci dessus, le premier pas est de prendre en compte les 8 bit de poids plus fort, qui a comme résultat la valeur 0x62.

Dans ce cas, comme on n'accepte qu'un adresse, la valeur de la masque sera 0x00 et du code le même que l'adresse tronqué, 0x62.

1.2 Accepter adresses: 0x310 et 0x320

Comme décrit ci dessus, le premier pas est de prendre en compte les 8 bit de poids plus fort, qui a comme résultat les valeur 0x62 et 0x64.

Pour le calcul de la masque on applique d'abord l'opération AND, en suite l'opération OR et finalement le XOR entre les deux résultats, qui nous donnent les valeurs 0x60, 0x66 et 0x06 respectivement. La valeurs de la masque est 0x06.

Le calcul du code se fait à partir de la masque et d'un des adresses. Le résultat est: 0x60.

Une remarque importante à faire c'est qui cette configuration n'ignore pas l'adresse 0x330 du au fait que l'ajout de cette adresse ne change pas les valeurs intermédiaires utilisés pour le calcul de la masque d'acceptation (0x60 et 0x66). Dans ce cas là une filtrage en logiciel est nécessaire.

1.3 Accepter adresses: 0x310, 0x320 et 0x340

Comme décrit ci dessus, le premier pas est de prendre en compte les 8 bit de poids plus fort, qui a comme résultat les valeur 0x62, 0x64 et 0x68.

Pour le calcul de la masque on applique d'abord l'opération AND, en suite l'opération OR et finalement le XOR entre les deux résultats, qui nous donnent les valeurs 0x60, 0x6E et 0x0E respectivement. La valeurs de la masque est 0x0E.

Le calcul du code se fait à partir de la masque et d'un des adresses. Le résultat est: 0x60.

Comme dans le cas précédant, une filtrage en logiciel en plus s'avère nécessaire vu que les adresses 0x330, 0x350, 0x360 et 0x370 ne sont pas filtrés une fois qu'ils sont une combinaison linéaire (terme à terme) des adresses cible (0x310, 0x320 et 0x340).

1.4 Accepter adresse: 0x370

Le calcul de la masque et du code d'acceptation est similaire à celui du premier exemple et donne les résultats 0x00 et 0x6E respectivement.

L'affichage de cette trame sur 2 octets à l'aide des LED's dont la carte est munie se trouve dans la vidéo suivante : `https://youtu.be/0MG2w1hz0Yg`.

2 Réception de données

Deux types différentes d'information on été envoyé qui devraient être décodée est affiché selon ses natures par la carte.

2.1 LED pattern

La première donnée était un pattern de LED codé sur 2 octets, mais avec seulement 10 bit d'information utile. Cet information était transmise pas let paquet avec l'identifient 0x370 donc pour ne recevoir que cette message, la filtrage décrite dans l'item 1.4 est nécessaire.

Le pattern reçu est celui illustré par l'image 1.

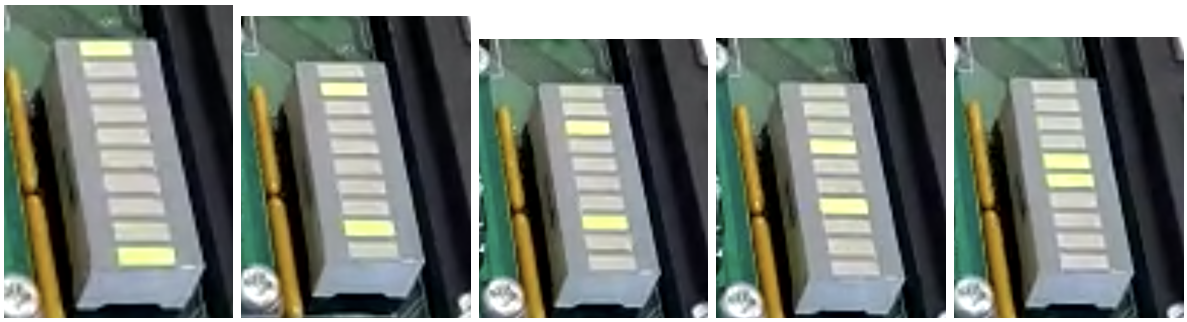

Figure 1: Pattern LED affiché après la réception des messages avec id 0x370

2.2 Message codée en plusieurs paquets

La deuxième information transmise était une message codée dans les paquets avec les id 0x310, 0x320, 0x330, 0x340, 0x350 et 0x360, donc la bonne réception de ces paquets nécessite la filtrage décrite dans

l'item 1.3 une fois qu'elle permet la réception de ces paquets. Une filtrage additionnelle est nécessaire pour enlever les paquets de la question précédente (id égal a 0x370).

Pour faire l'affichage du message contenu dans chaque paquet, il suffit d'afficher en console la valeur de chaque octet de donnée en format ASCII. Pour aider au développement et debuging, d'autres information comme l'adresse en 11 bits, la valeur du DLC et les données en format hexadécimal on était affichés comme on peut noter dans les images 2a et 2b. Dans la figure 2c on peut récupérer la message décodé complète.

```
address: 0x350   dlc: 0x7   led: 0x696C  data: 69 6C 20 6C 65 73 20
address: 0x360   dlc: 0x8   led: 0x4D32  data: 4D 32 20 50 72 6F 20 21
address: 0x310   dlc: 0x8   led: 0x4154  data: 41 54 54 45 4E 54 49 4F
address: 0x320   dlc: 0x8   led: 0x4E20  data: 4E 20 21 20 54 50 20 4E
address: 0x330   dlc: 0x7   led: 0x4F4E  data: 4F 4E 20 66 69 6E 69
address: 0x340   dlc: 0x8   led: 0x4175  data: 41 75 20 74 72 61 76 61
address: 0x350   dlc: 0x7   led: 0x696C  data: 69 6C 20 6C 65 73 20
address: 0x360   dlc: 0x8   led: 0x4D32  data: 4D 32 20 50 72 6F 20 21
address: 0x310   dlc: 0x8   led: 0x4154  data: 41 54 54 45 4E 54 49 4F
address: 0x320   dlc: 0x8   led: 0x4E20  data: 4E 20 21 20 54 50 20 4E
address: 0x330   dlc: 0x7   led: 0x4F4E  data: 4F 4E 20 66 69 6E 69
address: 0x340   dlc: 0x8   led: 0x4175  data: 41 75 20 74 72 61 76 61
address: 0x350   dlc: 0x7   led: 0x696C  data: 69 6C 20 6C 65 73 20
address: 0x360   dlc: 0x8   led: 0x4D32  data: 4D 32 20 50 72 6F 20 21
address: 0x310   dlc: 0x8   led: 0x4154  data: 41 54 54 45 4E 54 49 4F
address: 0x320   dlc: 0x8   led: 0x4E20  data: 4E 20 21 20 54 50 20 4E
address: 0x330   dlc: 0x7   led: 0x4F4E  data: 4F 4E 20 66 69 6E 69
address: 0x340   dlc: 0x8   led: 0x4175  data: 41 75 20 74 72 61 76 61
address: 0x350   dlc: 0x7   led: 0x696C  data: 69 6C 20 6C 65 73 20
address: 0x360   dlc: 0x8   led: 0x4D32  data: 4D 32 20 50 72 6F 20 21
address: 0x310   dlc: 0x8   led: 0x4154  data: 41 54 54 45 4E 54 49 4F
address: 0x320   dlc: 0x8   led: 0x4E20  data: 4E 20 21 20 54 50 20 4E
address: 0x330   dlc: 0x7   led: 0x4F4E  data: 4F 4E 20 66 69 6E 69
address: 0x340   dlc: 0x8   led: 0x4175  data: 41 75 20 74 72 61 76 61
address: 0x350   dlc: 0x7   led: 0x696C  data: 69 6C 20 6C 65 73 20
address: 0x360   dlc: 0x8   led: 0x4D32  data: 4D 32 20 50 72 6F 20 21
address: 0x310   dlc: 0x8   led: 0x4154  data: 41 54 54 45 4E 54 49 4F
address: 0x320   dlc: 0x8   led: 0x4E20  data: 4E 20 21 20 54 50 20 4E
address: 0x330   dlc: 0x7   led: 0x4F4E  data: 4F 4E 20 66 69 6E 69
address: 0x340   dlc: 0x8   led: 0x4175  data: 41 75 20 74 72 61 76 61
```

(a) Trame CAN avec données en Hexadécimal

```
address: 0x320   dlc: 0x8   led: 0x4E20  data: N   !   T P   N
address: 0x330   dlc: 0x7   led: 0x4F4E  data: O N   f i n i
address: 0x340   dlc: 0x8   led: 0x4175  data: A u   t r a v a
address: 0x350   dlc: 0x7   led: 0x696C  data: i l   l e s
address: 0x360   dlc: 0x8   led: 0x4D32  data: M 2   P r o   !
address: 0x310   dlc: 0x8   led: 0x4154  data: A T T E N T I O
address: 0x320   dlc: 0x8   led: 0x4E20  data: N   !   T P   N
address: 0x330   dlc: 0x7   led: 0x4F4E  data: O N   f i n i
address: 0x340   dlc: 0x8   led: 0x4175  data: A u   t r a v a
address: 0x350   dlc: 0x7   led: 0x696C  data: i l   l e s
address: 0x360   dlc: 0x8   led: 0x4D32  data: M 2   P r o   !
address: 0x310   dlc: 0x8   led: 0x4154  data: A T T E N T I O
address: 0x320   dlc: 0x8   led: 0x4E20  data: N   !   T P   N
address: 0x330   dlc: 0x7   led: 0x4F4E  data: O N   f i n i
address: 0x340   dlc: 0x8   led: 0x4175  data: A u   t r a v a
address: 0x350   dlc: 0x7   led: 0x696C  data: i l   l e s
address: 0x360   dlc: 0x8   led: 0x4D32  data: M 2   P r o   !
address: 0x310   dlc: 0x8   led: 0x4154  data: A T T E N T I O
address: 0x320   dlc: 0x8   led: 0x4E20  data: N   !   T P   N
address: 0x330   dlc: 0x7   led: 0x4F4E  data: O N   f i n i
address: 0x340   dlc: 0x8   led: 0x4175  data: A u   t r a v a
address: 0x350   dlc: 0x7   led: 0x696C  data: i l   l e s
address: 0x360   dlc: 0x8   led: 0x4D32  data: M 2   P r o   !
address: 0x310   dlc: 0x8   led: 0x4154  data: A T T E N T I O
address: 0x320   dlc: 0x8   led: 0x4E20  data: N   !   T P   N
address: 0x330   dlc: 0x7   led: 0x4F4E  data: O N   f i n i
address: 0x340   dlc: 0x8   led: 0x4175  data: A u   t r a v a
address: 0x350   dlc: 0x7   led: 0x696C  data: i l   l e s
address: 0x360   dlc: 0x8   led: 0x4D32  data: M 2   P r o   !
address: 0x310   dlc: 0x8   led: 0x4154  data: A T T E N T I O
```

(b) Trame CAN avec données en ASCII

```
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
ATTENTIO N ! TP N ON fini Au trava il les  M2 Pro !
```

(c) Message ordonné

Figure 2: Décodage du message

3 Envoi d'un message

Pour cela on reprend la solution de temporisation matérielle développé lors du premier TP afin de respecter la consigne de temporisation. Pour avoir une transmission de données avec une période de 800 ms, on a besoin d'un compteur qui compte jusqu'à 4162, vu que chaque le période d'interruption est de approximativement 192 μs. Une fois cette valeur est atteinte, le compteur est remis à zéro et la flag FLAG_tempo est mise à 1.

Après cela, l'envoie du message se fait à partir de la fonction *send_data* qui prend un vecteur d'octets de taille inférieure à 30. L'envoie est fait en plusieurs paquets si la taille des données dépasse 8 octets, vu que chaque paquet n'envoie que 8 octets de donnée utile.

Pour garantir la bonne réception de tous les parties de la donnée originale, les 3 derniers bits de l'adresse sont utilisés pour identifier chaque partie vu que la filtrage matérielle les ignore. De cette façon, le premier paquet a l'id 0x490, l'adresse de base, et les 8 premiers octets de la donnée, le deuxième l'id 0x491 et les 8 octets de donnée suivants jusqu'au quatrième qui a l'id 0x493 et les dernier 5 octets de donnée.

Théoriquement l'espace de sous-id décrit ici permet jusqu'à 8 pacquets, qui résulte en 64 octets de données utiles, mais dans le cas de ce TP cette limite est définie comme 29.