

Hardware Software Interface

Projet (par groupes de 2 ou 3 élèves)

Noté sur 15

Alexis DALEY

Sujet

Nous allons coder le logiciel applicatif du calculateur principal d'une voiture pas à pas.

Les objectifs sont les suivants :

- Maîtriser les bonnes pratiques à appliquer lors du développement d'un système embarqué temps réel.
- Respecter et assurer le fonctionnement déterministe du système.
- Maîtriser le développement d'une machine à état fini.

Contraintes et conseils

- Vous êtes libres de créer autant de fichiers .c/.h que vous le souhaitez, excepté lorsque le sujet vous oriente sur ce point.
- Vous êtes libres de définir l'arborescence de projet qui vous convient le mieux, excepté lorsque le sujet vous oriente sur ce point.
- Pensez, si nécessaire, à bien définir vos données en utilisant les types : structures, énumérés et union.
- Privilégiez l'utilisation des #define afin d'éviter d'utiliser des valeurs en dure dans vos fonctions.
- Pensez à ajouter des commentaires à vos fichiers, fonctions, types, et opérations logiques complexes.

La qualité et clarté du code sera notée sur 2 points.

Détails du sujet

Le BCGV (Boitier Central de Gestion de la Voiture) est responsable du bon fonctionnement du système global ([voir annexe 3](#)) : la voiture. Le BCGV est en charge d'assurer les fonctionnalités suivantes décrites dans le document de spécification :

- Interagir avec les équipements suivants : Tableau de bord, commodos, Feux, Moteur, Batterie et Châssis
- Recevoir et traiter les contrôles des feux et des essuis glaces de l'utilisateur
- Recevoir et traiter les problèmes et mesures des autres équipements telles que les problèmes moteurs, le kilométrage, le niveau du réservoir, ou la charge batterie
- Contrôler les différents feux de la voiture
- Contrôler les voyants du tableau de bord de la voiture

Lors de ce TP vous développerez seulement le logiciel applicatif du BCGV. Le logiciel « bas niveau » BSP (board support package) contenant entre autres les drivers et permettant d'interagir avec les autres systèmes (Tableau de bord, commodos, Feux, Moteur, Batterie et Châssis) vous est fourni sous la forme d'un exécutable « *driver* », d'une librairie statique « *drv_api.a* » et de son fichier .h « *drv_api.h* ».

Le logiciel que vous développerez devra interagir avec les sous-systèmes en utilisant cette interface. Ces API vous permettront de lire et écrire des données sur les différentes liaisons (UDP, Série) de votre ordinateur. Derrière cette librairie statique se cache des appels à des sockets qui communiquent avec le processus du driver.

(Pour les curieux : <https://broux.developpez.com/articles/c/sockets/>)

Avant de démarrer, prenez bien le temps de lire toutes les questions, et surtout de lire l'[Annexe 1](#) sur les règles de codage imposées

Question 1.

En utilisant l'[Annexe 2](#), l'[Annexe 3](#), les tableaux des **Question 6 et 7** décrivant le fonctionnement de votre ordinateur, faire un document (excel, csv, json...) qui va définir vos données applicatives. Chaque donnée devrait être d'un type créé par vous, qui peut référer à un type primitif (typedef uint32_t mon_type_t) Votre document doit contenir 2 tableaux avec les colonnes suivantes :

Types

Nom, Genre (Enum ou Atom*), déclaration (ex : uint32_t, ou déclaration de l'enum),
Domaine, commentaire

Données

Id, Nom, Type, Dimension, Valeur d'init, commentaire

* : Un type atomique sera un type C ou un type de stdint.h

Question 2.

Faire une librairie statique (.a) contenant :

- La définition de tous les types
- La définition de la structure de données applicatives
- Son instantiation
- Un getter pour chaque donnée
- Un setter pour chaque donnée qui vérifie les bornes de la donnée avant de changer sa valeur

Faire un script dans le langage de votre choix pour générer ce code automatiquement depuis votre document créé en *Question 1*.

Question 3.

Il est temps de faire un *main* !

Faites la fonction *main* de votre application dans le fichier *app.c*, qui lit et affiche la trame envoyée en UDP toutes les 10 millisecondes par le calculateur MUX.

Cette trame est bloquante. Ce qui veut dire que l'appel de l'api va vous bloquer jusqu'à recevoir cette trame. Elle agit donc comme un vrai séquenceur pour le soft de votre calculateur. Même si ce n'est pas forcément le cas, votre soft se basera sur le fait que cette trame est périodique à 10ms pour faire toutes les mesures de temps.

Votre exécutable devra s'appeler *app*.

Question 4.

En utilisant le fichier *fsm.c* (finite state machine) comme base, codez chacune des machines à états de l'[annexe 2](#). Chacune de vos state machine aura une fonction qui sera exécutée toutes les 10ms par le *main*.

Vous devez donc adapter ces fichiers pour les rendre utilisables dans votre soft :

- Supprimer les états, fonctions de transitions, et le tableau de transition pour les remplacer par les vôtres
- Ajouter vos états, vos fonctions de callback (si besoin), et votre tableau de transitions pour chaque fsm
- Modifier la fonction *GetNextEvent*, qui permet de lever l'événement en cours. Elle devra aller chercher les paramètres de votre FSM et définir un événement.
- N'oubliez pas les commentaires doxygen pour vos fonctions et vos fichiers 😊
- Adapter le *main* pour en faire une fonction appellable toutes les 10ms par votre *main* plutôt qu'un *main* qui tourne (ce qu'il y a dans la boucle « for » ne doit presque pas bouger).
- Faire un fichier *fsm.h* pour chaque state machine qui met à disposition la fonction à appeler par votre application.

Assurez-vous ensuite de coder les exigences écrites spécifiées dans cette la même Annexe.

(Explication orale du professeur requise)

Question 5.

Créez un Makefile pour compiler votre application et un autre pour votre librairie statique.

Créez un Makefile « maître » permettant d'appeler les deux en même temps

(Aide : <https://gl.developpez.com/tutoriel/outil/makefile/>)

Faites un fichier bash qui lance le driver et votre applicatif.

Question 6.

Créez une fonction de décodage, mettant à jour les données de votre structure applicative, pour chaque trame reçue par le driver selon les informations ci-dessous :

- COMODO -> BCGV (LNS périodique 100ms, Little endian, 2 octets)

MSB

LSB

Taille (bits)	1	1	1	1	1	1	1	1	8
Info	Cmd. warning	Cmd. feux de position	Cmd. feux de croisement	Cmd. feux de route	Cmd. clignotant droit	Cmd. clignotant gauche	Cmd. essuie-glaces	Cmd. lave glace	CRC8
Valeur	booléen	booléen	booléen	booléen	booléen	booléen	booléen	booléen	Calculé sur le reste de la trame

Calcul du CRC8 : <https://github.com/lammertb/libcrc/blob/master/src/crc8.c>

- MUX -> BCGV (UDP périodique 10ms, Big endian, 14 octets)

MSB

LSB

Taille (octets)	1	4	1	1	1	1	4	1
Info	Numéro de trame	Kilométrage	Vitesse	Problème Châssis	Problèmes Moteur	Niveau Réservoir	Tours/minutes	Problèmes Batterie
Équipement	MUX	Châssis	Châssis	Châssis	Moteur	Moteur	Moteur	Batterie
Valeur	Entier non signé 0<n<100 Rebouclage 100->1	Entier non signé (km)	Entier non signé (km/h) Max : 300	(Porteur de bits) 0x0 : Aucun 0x1 : Pression pneus 0x2 : freins usés	(Porteur de bits) 0x0: Aucun 0x1: Default pression 0x2: Temperature LDR* 0x4: Surchauffe huile	Entier non signé (Litres) Réservoir 40L	Entier non signé (Tours/minute) Max : 10 000	(Porteur de bits) 0x0: Aucun 0x1: Déchargée 0x2: Panne

*: LDR : Liquide de refroidissement

Question 7.

Créez une fonction d'encodage chaque trame ci-dessous, utilisant les valeurs de vos données pour former les messages :

- BCGV -> BGF 1 (asynchrone, 2 octets)

Taille (octet)	1	1
Info	Id de message	Activation feux de position
Valeur	Entier non signé 0x01	0x0 : Éteints 0x1 : Allumés

- BCGV -> BGF 2 (asynchrone, 2 octets)

Taille (octet)	1	1
Info	Id de message	Activation feux de croisement
Valeur	Entier non signé 0x02	0x0 : Éteints 0x1 : Allumés

- BCGV -> BGF 3 (asynchrone, 2 octets)

Taille (octet)	1	1
Info	Id de message	Activation feux de route
Valeur	Entier non signé 0x03	0x0 : Éteints 0x1 : Allumés

- BCGV -> BGF 4 (asynchrone, 2 octets)

Taille (octet)	1	1
Info	Id de message	Activation feux clignotant droit
Valeur	Entier non signé 0x04	0x0 : Éteints 0x1 : Allumés

- BCGV -> BGF 5 (asynchrone, 2 octets)

Taille (octet)	1	1
Info	Id de message	Activation feux clignotant gauche
Valeur	Entier non signé 0x05	0x0 : Éteints 0x1 : Allumés

En réponse de chacun des messages ci-dessus, le BGF répond par un message d'acquittement (qui signifie : « bien reçu ! ») contenant exactement les mêmes valeurs que le message que vous avez envoyé.

A partir de la réception des messages d'acquittements seulement, vous pouvez considérer que votre action a bien été prise en compte.

- BCGV->MUX (UDP périodique 20ms, Big endian, 9 octets)

MSB

Taille (bits)	1	1	1	1	1	1	1	1
Info	Voyant feux de position	Voyant feux de croisement	Voyant feux de route	Voyant essence	Voyant défaut moteur	Voyant pression pneus	Voyant Plaquettes freins usés	Voyant batterie déchargée
Équipement	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord
Valeur	booléen	booléen	booléen	booléen	booléen	booléen	booléen	booléen

LSB

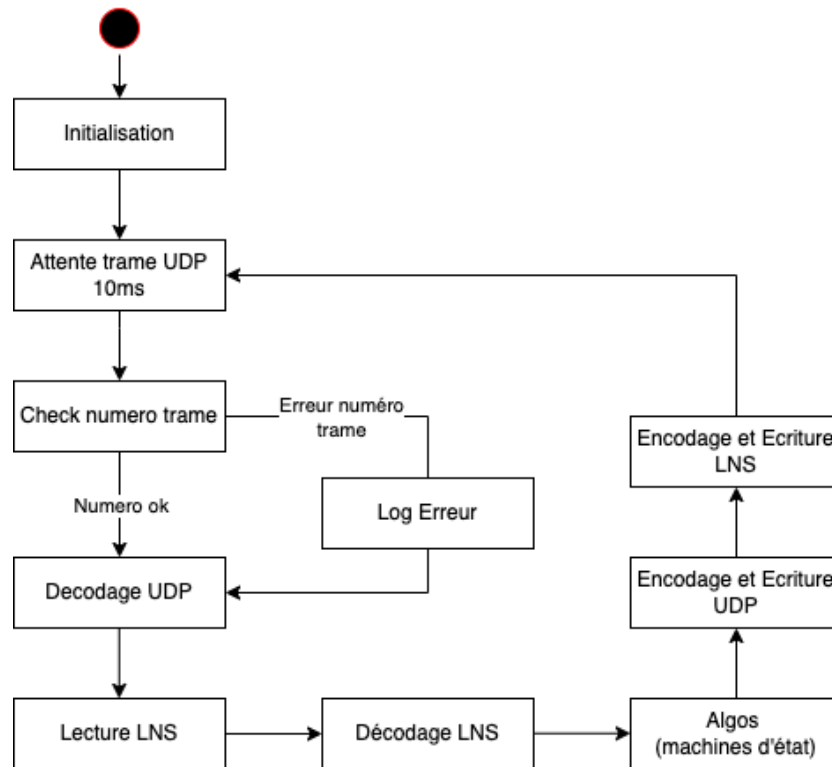
Taille (bits)	1	1	1	1	1	1	1	1
Info	Voyant Warnings	Voyant Panne batterie	Voyant Temp. LDR*	Voyant pression moteur	Voyant surchauffe huile	Voyant défaillance freins	Activation Essuie-glaces	Activation Lave-glace
Équipement	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord	Pare-Brise	Pare-Brise
Valeur	booléen	booléen	booléen	booléen	booléen	booléen	booléen	booléen

*: LDR : Liquide de refroidissement

Taille (bits)	32	8	8	8
Info	Affichage kilométrage	Affichage vitesse	Affichage niveau réservoir	Affichage tours/minutes
Équipement	Tableau de bord	Tableau de bord	Tableau de bord	Tableau de bord
Valeur	Entier non signé 4 octets (km)	Entier non signé (km/h)	Entier non signé (%)	Entier non signé (tours/min)

Question 8.

Mettez à jour votre main pour convenir au diagramme ci-dessous :



La trame UPD contient un numéro de trame qui s'incrémente entre deux trames. L'appel étant bloquant (on attend la trame suivante pour se débloquent), il se peut que votre soft loupe une trame si jamais le reste du traitement est trop long. Il faut donc vérifier que ce numéro est bon. Si ce n'est pas le cas, traitez la trame reçue et laissez un log (printf).

(Explication du professeur pour les questions 9,10,11)

Question 9.

Pour être sûr de ne jamais louper de trame UDP, créez un thread (créé au lancement de votre application), qui s'occupe de lire les données du driver et de les enregistrer dans un buffer.

Pour protéger votre structure applicative des accès concurrentiels à la mémoire entre vos 2 threads, protégez votre structure par un mutex. (Utilisez les fonctions `pthread_mutex_lock` et `pthread_mutex_unlock`).

Votre thread main doit quand même être cadencé à 10ms comme avant. Pour cela vous pouvez utiliser `#include <time.h>` et la fonction `clock()`.

Question 10.

Faire un diagramme de séquence avec le logiciel de votre choix (<https://www.diagrams.net/> est très bon si vous n'en avez pas), représentant le comportement de votre système avec les acteurs suivants : App, Struct (structure applicative), thread lecteur, driver.

Que pensez-vous de la performance de votre soft entre avant et après la création de votre deuxième thread ? Identifiez-vous des problèmes ? Pensez-vous que votre soft est déterministe ?

Question 11.

Remplacez votre buffer par la fifo fournie (fichier `fifo.c/.h`), qui permet à deux threads de s'échanger des données **sans mutex**.

Mettez à jour votre Diagramme de séquence de la question 10.

Annexe 1 : Règles de codage

- Ne mélangez pas l'anglais et le français dans votre code et vos commentaires
Si vous faites du français, utilisez-le partout. Pas de franglais.
- Pour le nommage, pareil, pas de mélange. Utiliser camel ou snake case, mais ne mélangez pas les deux dans vos variables, ni dans vos fonctions.
(En revanche vous avez le droit d'utiliser snake pour vos fonctions, et camel pour vos variables par ex).
- De manière générale, veillez à ne pas faire de noms (fonctions, variables) à rallonge.
- Les types custom doivent commencer par 't' ou finir par '_t'
- Pour les defines et macros : majuscules et avec des underscore "_" pour séparer les mots
Ex : `#define MON_DEFINE (1)`
- Pour les entiers, vous devez utiliser ceux de la lib `<stdint.h>` (uint32_t, int32_t etc.)
- Fonction malloc interdite
- Compilation avec **-W -Wall -pedantic** imposée. Aucun warning accepté
- Commentaires doxygen de fichier :

```
/**  
 * \file main.c  
 * \brief Description breve  
 * \details Details ou notes d'implementation (peut rester vide)  
 * \author Noms  
 */
```

- Commentaires doxygen de fonction (si la fonction est déclarée dans le .h, mettre les commentaires que dans le .h) :

```
/**  
 * \brief Description breve de la fonction.  
 * \details Details ou notes d'implementation (peut rester vide)  
 * \param param1 : Description  
 * \param param2 : Description  
 * \return int32_t : Description  
 */  
int32_t fonction(uint8_t param1, uint32_t param1);
```


- Vous pouvez choisir des règles différentes de celles utilisées dans les fichiers fournis, ou les extraits de codes de ce document.

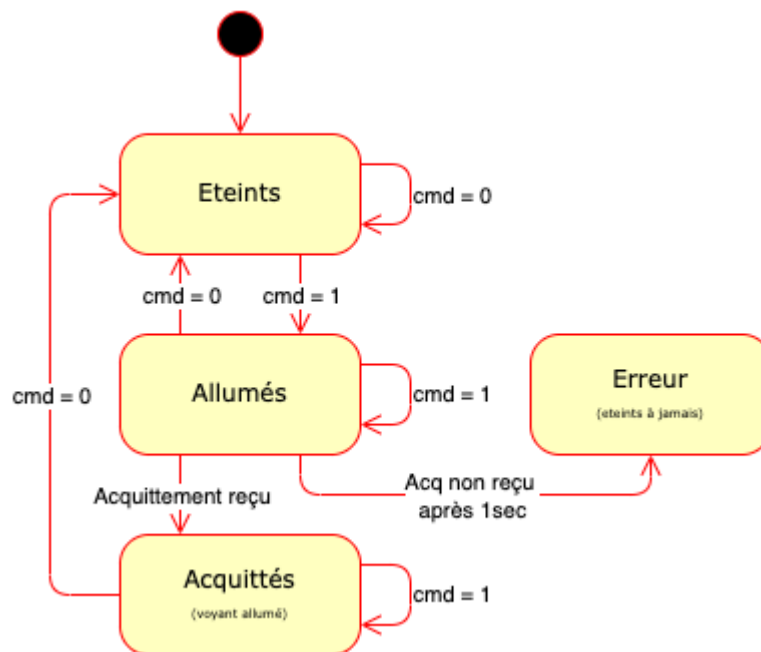
Annexe 2 : Exigences du système

Ci-dessous les exigences de votre système.

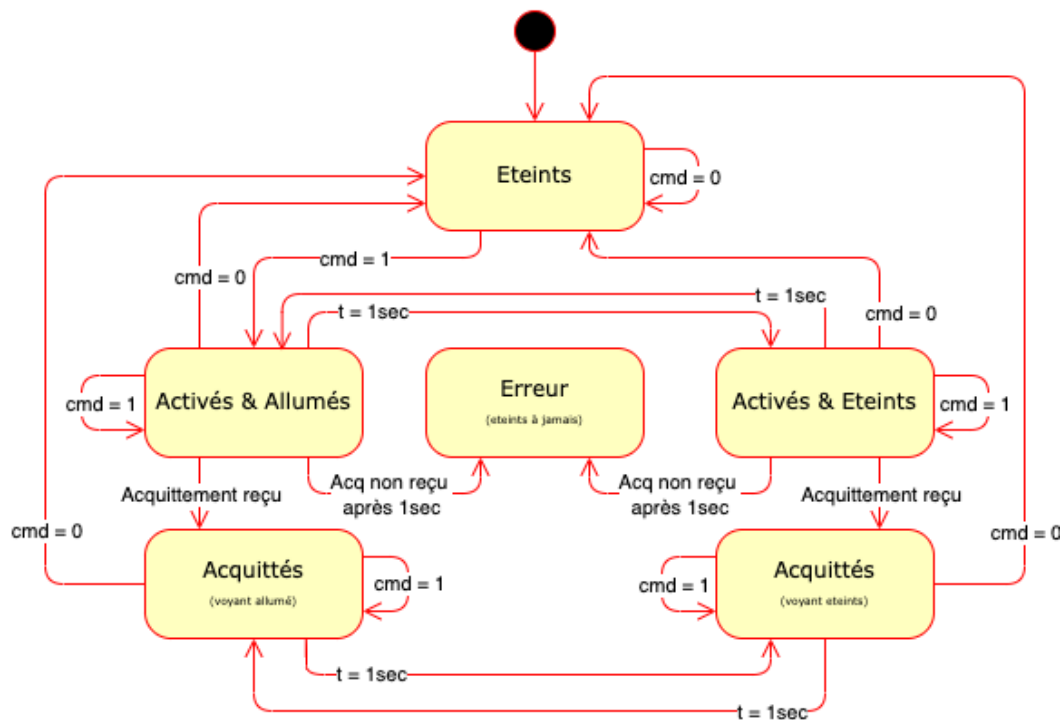
Cmd = commande

State machine des feux classiques

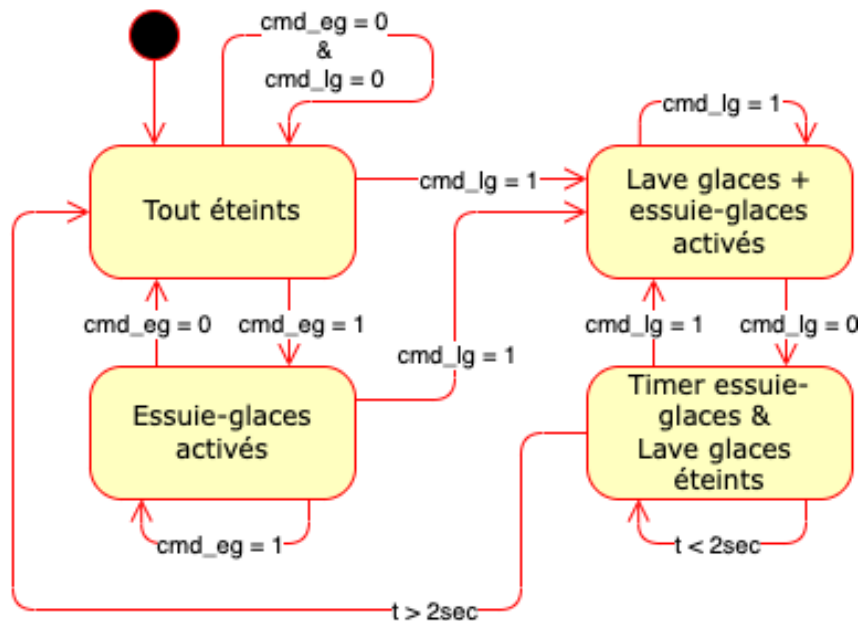
(position/croisement/route)



State machine des feux clignotants et warnings



State machine des essuie-glaces (eg) et lave glace (lg)



Exigences du BGCV :

- Le BCGV doit répondre aux 3 machines à états ci-dessus, concernant les éléments suivants :
 - Gestion de l'allumage et de l'arrêt des feux de position
 - Gestion de l'allumage et de l'arrêt des feux de croisement
 - Gestion de l'allumage et de l'arrêt des feux route
 - Gestion de l'allumage et de l'arrêt des feux de détresse (warnings)
 - Gestion de l'allumage et de l'arrêt des feux clignotants droits et gauche
 - Gestion de l'allumage et de l'arrêt des essuie-glaces et du lave-glace

- Le BCGV doit réceptionner toutes les 10ms la trame UDP envoyée par le MUX. Si ce n'est pas le cas, un log sera fait et le soft continuera de tourner.
- Les valeurs de Vitesse, kilométrage, tours/minutes et niveau de réservoir reçue en UDP par les différents équipements doivent être renvoyées en UDP dans la trame 20ms, pour affichage sur le tableau de bord.
- Le BCGV ne doit pas décoder la trame COMODO -> BCGV seulement si le CRC8 est valide
- Le BCGV doit gérer l'allumage ou non des voyants suivants :
 - Feux de position, croisement, route
 - Clignotants gauches et droits, feux de détresse (warnings)
 - Essence (réservoir rempli à 5% ou moins)
 - Défaut moteur
 - Pression des pneus
 - Plaquettes de freins usées
 - Batterie déchargée, Panne Batterie
 - Température liquide de refroidissement, pression moteur et surchauffe huile moteur
 - Défaillance des freins
 - Activation lave-glace et essuie-glaces

Annexe 3 : Architecture Hardware du système

Le calculateur que vous codez est le **BCGV**

