



Bureau d'étude Electronique Automobile



http://lesia.insa-toulouse.fr/~a_boyer

Alexandre Boyer
Patrick Tounsi

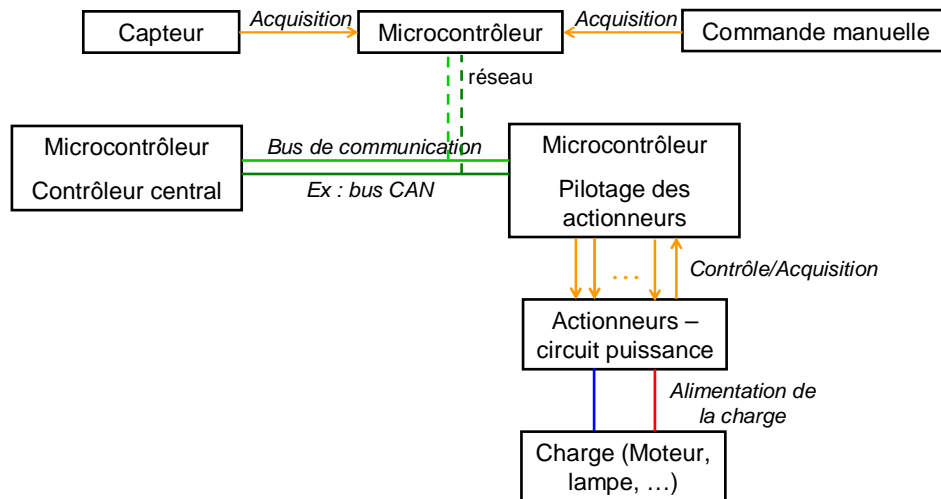
5^e année ESE
Octobre 2010

Contenu

I -	Contexte	3
II -	Objectifs du bureau d'étude	5
III -	Enoncé du BE – Cahier des charges.....	6
	1. Enoncé.....	6
	2. Cahier des charges - Exigences clients.....	7
	a. Exigences fonctionnelles.....	7
	b. Exigences sur le BCM.....	9
	c. Exigences sur la communication CAN	9
	d. Exigences sur la gestion de l'énergie	10
	e. Exigences en terme de diagnostic des fautes.....	11
IV -	Organisation et Planning	12
V -	Notation.....	12
VI -	Annexe 1 – Format des documents de spécifications	14
VII -	Annexe 2 – Présentation du matériel.....	17
	1. Les maquettes.....	17
	a. Module portière	17
	b. Module volant	17
	2. Les cartes électroniques	18
	a. MC33887 – Pont en H.....	18
	b. MC33984 – Dual High side switch	18
	c. MC33981 – Single High Side driver.....	19
	d. Kit de développement DEMO9S12XEP100	20
	3. Carte d'interface	22
	4. Carte volant	23
	5. Carte afficheur LCD.....	23
VIII -	Annexe 3 - Prise en main du matériel	25
	1. Prise en main de la carte MC33887	25
	2. Prise en main de la carte MC33984	25
	3. Prise en main de la carte de développement DEMO9S12XEP100 et de l'outil de programmation Freescale CodeWarrior v4.7	25

I - Contexte

Dans les véhicules actuels sont embarqués de nombreux composants électroniques (microcontrôleur, capteur, actionneur de puissance, ...) permettant d'améliorer la fiabilité du système, la sécurité et le confort des passagers et le rendement énergétique. La mise en place et l'optimisation de tous ces organes électroniques à l'intérieur d'un véhicule nécessite un savoir faire large en électronique (analogique, numérique, puissance) et en informatique matérielle. La figure ci-dessous décrit l'architecture typique d'une application automobile.



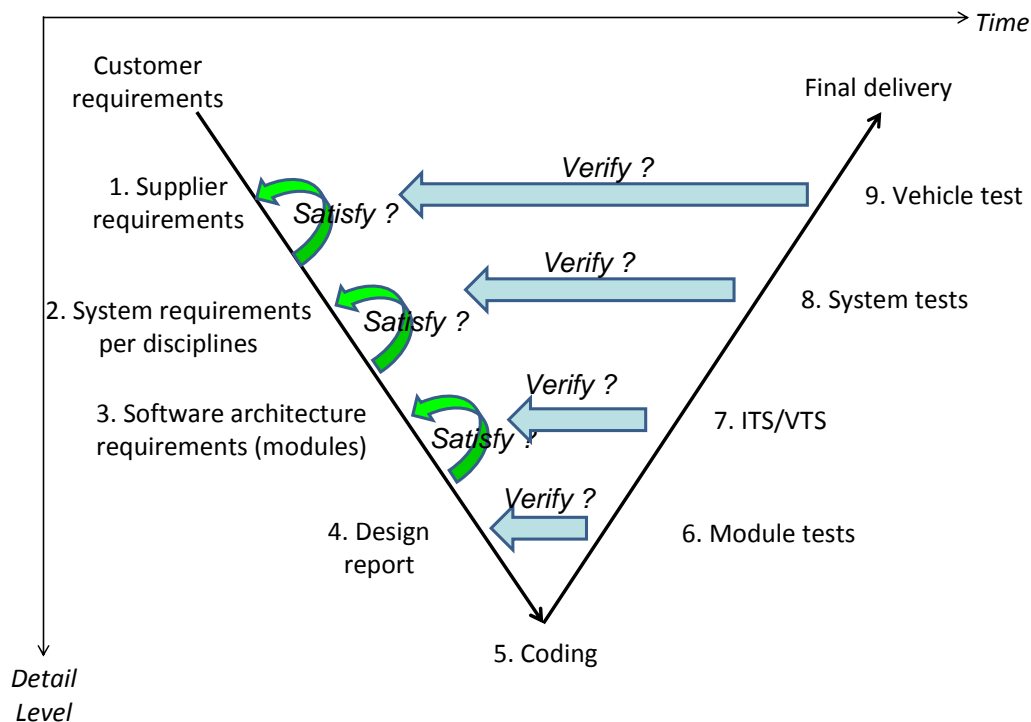
Une charge (lampe, moteur) est pilotée par un actionneur capable de délivrer le courant nominal absorbé par la charge. Ce circuit de puissance est lui-même commandé par un microcontrôleur. Les circuits de puissance utilisés dans l'industrie automobile ne sont pas de simples commutateurs de puissances puisqu'ils intègrent aussi toute une partie logique de contrôle, disposent de plusieurs modes de fonctionnement et renvoient de nombreuses informations comme la température ou le courant débité. L'actionnement des charges dépend des informations acquises et envoyées par un ensemble de capteurs aux organes de contrôle. Tous ces organes de contrôle forment un réseau interconnecté par un ou plusieurs bus, comme le bus standard CAN.

En raison des exigences économiques, de sûreté de fonctionnement et de robustesse, le processus de conception d'une application automobile suit souvent un cycle particulier, appelé cycle en V, décrit à la figure ci-dessous.

Ce cycle de conception a pour objectifs de faciliter l'élaboration d'un produit final à partir de spécifications client, vérifier la cohérence et le respect des spécifications client à chaque étape et le « debug » des problèmes. Dans le cadre du développement d'une application électronique automobile, les différentes étapes sont :

1. Exigences équipementiers : A partir des exigences ou spécifications client, l'équipementier définit son propre cahier des charges. Le cahier des charges contient des exigences fonctionnelles et des contraintes auxquelles doit répondre l'application.
2. Exigences systèmes par discipline : les exigences en différentes disciplines : logicielle (code embarqué), électronique (conception carte et choix composants) et mécanique. Il est nécessaire de s'assurer que les exigences systèmes doivent se conformer aux exigences de la phase 1. Dans le cadre de ce BE, nous mettrons l'accent sur la

conception du code embarqué en fonction des exigences électroniques et fonctionnelles du système.



3. Exigences d'architecture logicielle : l'analyse des exigences logicielles permet de proposer l'architecture de l'application logicielle, décomposée en modules. Les exigences logicielles sont décrites dans un rapport. Il est nécessaire de s'assurer que les exigences systèmes se conforment aux exigences de la phase précédente.
4. Exigence design : Chaque module est décomposé en fonctions. Il est nécessaire de s'assurer que les exigences systèmes se conforment aux exigences de la phase 3. Les tests modulaires peuvent être définis à cette étape (pas leur mise en œuvre).
5. Codage : Il s'agit de l'écriture proprement dite du code permettant de satisfaire aux exigences de design. Le codage est soumis à un ensemble de recommandations et de contraintes afin d'en améliorer la portabilité, la lisibilité, la robustesse ...
6. Tests modulaires : Chaque fonction doit être testée et validée. Des vecteurs de tests sont choisis afin d'obtenir une couverture de test = 100 % et garantir le bon fonctionnement de l'application finale. Un rapport est créé qui détaille le test des fonctions. On s'assure que toutes les exigences définies en 4 sont respectées.
7. Integration Test Specification (ITS) / Verification Test Specification (VTS) : Le test ITS permet de s'assurer la bonne compatibilité entre les modules, leur initialisation correcte, leur bonne communication, la bonne récurrence des fonctions dans le temps. Le test VTS permet de s'assurer que les exigences sont respectées. On s'assure qu'on respecte toutes les exigences définies en 3.
8. Tests système : On vérifie sur l'équipement l'ensemble du code. On s'assure qu'on respecte toutes les exigences définies en 2.

9. Tests véhicule : il s'agit du test final avant le livrable pour le client.

L'organisation de ce bureau d'étude suivra ce type de cycle de conception, dans la mesure du temps et du matériel disponible. Le travail réalisé durant les 9 séances de BE s'inscrit dans les étapes 3, 4, 5, 6 et 7 de ce cycle de conception.

II - Objectifs du bureau d'étude

Le but de ce bureau d'étude est de réaliser une application automobile basé sur l'architecture précédente en suivant un processus de développement industriel. L'application logicielle (le code embarqué) à réaliser est spécifiée par un cahier des charges définissant non seulement les exigences fonctionnelles, mais aussi les exigences en termes de gestion d'énergie, de robustesse du système aux erreurs et aux pannes, de sécurité du passager.

Pour cela, vous disposez d'un ensemble de composants électroniques dédiés aux applications automobiles, fournis par la société Freescale Semiconductor, et des logiciels de programmation associés (Freescale CodeWarrior).

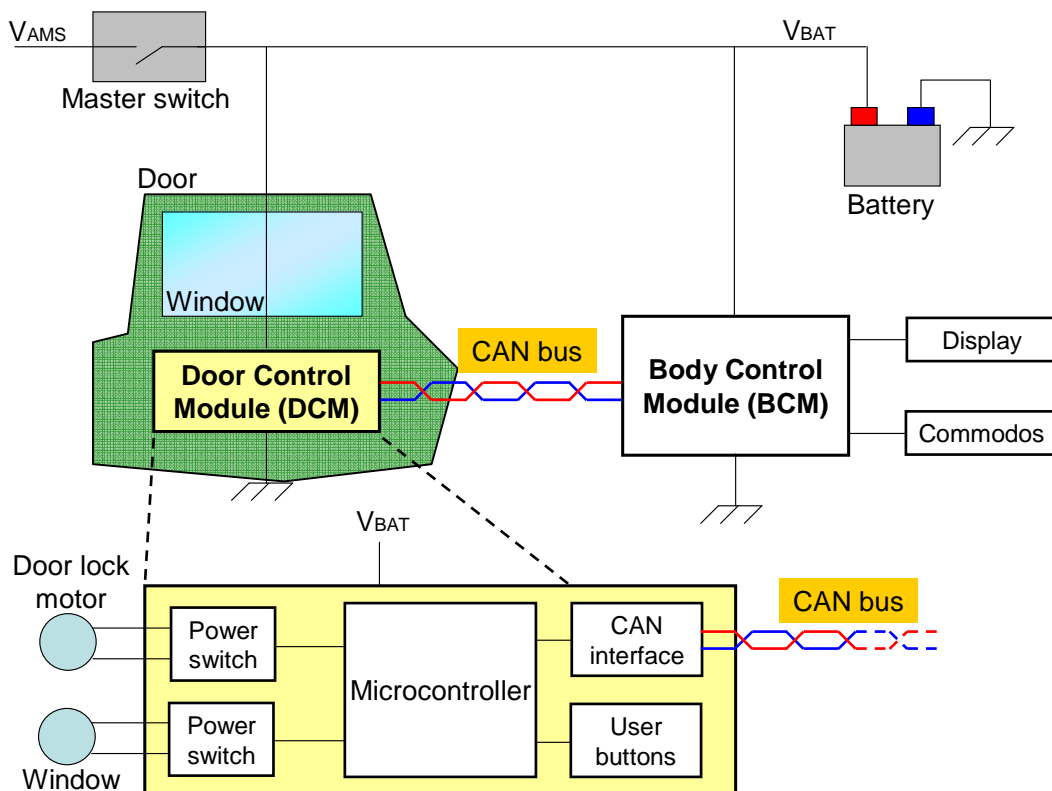
Les objectifs du bureau d'étude sont les suivants :

- mettre en place une architecture typique d'une application automobile (microcontrôleur, actionneur, charge, bus)
- développer un projet complet : mise en place du cahier des charges, développement logiciel et mise en œuvre pratique à l'aide de composants électroniques
- proposer des solutions logicielles et matérielles améliorant la sécurité, le confort du conducteur et des passagers, réduisant la consommation d'énergie, le risque d'erreurs matérielles et logicielles et permettant le diagnostic du système
- se familiariser aux composants industriels automobiles et aux contraintes de ce domaine
- mettre en réseau l'ensemble des composants en utilisant un bus de terrain tel que le bus CAN (Controller Area Network)

III - Enoncé du BE – Cahier des charges

1. Enoncé

Votre équipe est en charge du développement d'un nouvel ECU (Electronic Control Unit) embarqué dans les futurs véhicules de vos clients. Cet ECU, appelé Door Control Module (DCM) est en charge de la gestion des fonctions associées aux portières du véhicule. Il gère l'ouverture/la fermeture des portières et la montée/descente des vitres. Le module DCM est esclave du contrôleur central d'habitacle, appelé Body Controller Module (BCM), avec lequel il communique à l'aide d'une liaison par bus CAN (Controller Area Network). La figure ci-dessous décrit l'architecture matérielle de l'application et l'interconnexion entre les différents équipements.



Votre mission est développer l'application logicielle de gestion de la portière, embarquée dans le DCM et le BCM. Cette application doit respecter les exigences du client définies dans le cahier des charges ci-dessous. Votre travail se décompose en différentes parties :

1. Ecrire la spécification de l'architecture matérielle et logicielle de l'application
2. Ecrire la spécification logicielle par module
3. Coder l'application
4. Définir des vecteurs de tests, tester et valider chaque module
5. Tester et valider l'application complète

Même s'il s'agit d'un travail en binôme, le travail sur la spécification de l'application et le test final est un travail de groupe. Le travail par binôme concerne la spécification détaillée des modules et leur codage.

Des livrables sont associés aux 5 parties définies ci-dessus (Se reporter à IV. Planning et à V. Notation pour plus de détails sur les livrables). Les livrables de spécification devront suivre le format décrit dans l'annexe 1. Pour développer cette application, un module volant, un module portière et un ensemble de cartes de développement vous sont mis à disposition (cf. annexe 2).

2. Cahier des charges - Exigences clients

a. Exigences fonctionnelles

Description de la fonction « Verrouillage Portière »

La commande du verrouillage (ou déverrouillage) de la portière est assurée par :

- un bouton poussoir situé sur la portière, actionné par un conducteur ou un passager.
- un interrupteur placé sur le tableau de bord, actionné par un conducteur ou un passager. Celui-ci est géré par le module BCM.
- par le RKE (Remote Key Entry), géré par le module BCM.
- si la vitesse du véhicule dépasse les 10 km/h, la portière se verrouille. L'information de vitesse véhicule est récupérée par le BCM.

La commande du moteur du verrouillage de la portière est assurée par un switch de puissance commandé par une impulsion. **Le diagnostic du switch de puissance par le contrôleur DCM permet de vérifier si l'action de verrouillage/déverrouillage s'est correctement effectuée.**

A l'initialisation du véhicule, on suppose que la porte est déverrouillée. Le verrouillage/déverrouillage de la portière doit être effectué en moins de 200 ms. L'état de verrouillage de la portière est connu par le DCM et par le BCM, quel que soit l'origine de l'ordre de verrouillage.

Le verrouillage/déverrouillage des portières est prioritaire devant la fonction lève vitre. Une opération de verrouillage/déverrouillage en cours ne peut pas être annulée.

Scénario dans le cas où l'ordre de verrouillage/déverrouillage provient de la portière :

Le DCM effectue l'opération de verrouillage/déverrouillage puis avertit le BCM du changement d'état de la portière en lui transmettant par bus CAN l'état de verrouillage. Si le message a été bien reçu par le BCM, celui-ci doit transmettre un acquittement.

Si, en raison d'une erreur de transmission, le BCM n'est pas en accord avec l'état de la portière, le BCM modifie l'état de verrouillage de la portière qu'il a en mémoire, sans prévenir le DCM.

Si le DCM ne reçoit pas l'acquiescement du BCM au bout de 200 ms, il relance la demande d'acquiescement. En cas d'absence de réponse, il renouvelle 4 fois la demande. Au bout de 5 demandes sans réponses, le DCM abandonne la demande d'acquiescement.

Scénario dans le cas où l'ordre de verrouillage/déverrouillage provient du BCM :

Le BCM transmet l'ordre de verrouillage/déverrouillage de la portière et demande au DCM d'acquiescer l'opération en lui renvoyant le nouvel état de verrouillage de la portière. Si le DCM reçoit correctement le message, il effectue l'opération nécessaire puis s'acquiesce de l'action. Si le BCM transmet un ordre erroné (par exemple, le BCM demande de fermer la

portière alors qu'elle est déjà fermée), le DCM n'effectue aucune opération de verrouillage, mais il envoie quand même un message d'acquiescement au BCM.

Si le DCM ne répond pas au bout de 200 ms, le BCM renvoie l'ordre de verrouillage/déverrouillage. En cas d'absence de réponse, il renouvelle 4 fois la demande. Au bout de 5 demandes sans réponses, le BCM abandonne la demande d'acquiescement, affiche une erreur sur le tableau de bord et conserve en mémoire l'état de verrouillage.

Description de la fonction lève vitre

La commande lève vitre est assurée par :

- deux boutons poussoirs situés sur la portière (on utilisera les boutons poussoirs disponibles sur la carte de développement DEMO9S12XEP100), actionnés par un conducteur ou un passager. Le premier commande la montée du lève vitre, le second la descente. En cas d'appui simultané, la fonction est inhibée. Dans le cas d'un appui long (> 100 ms), la montée/descente de la vitre se fera en mode manuel (l'action dure tant que l'utilisateur appuie sur le bouton). Dans le cas d'un appui court (< 100 ms), la montée/descente de la vitre se fera en mode automatique (l'action dure tant que l'utilisateur n'appuie pas à nouveau sur un des boutons poussoirs).
- Si le véhicule n'est pas mis en marche et si de la pluie est détectée, les vitres doivent remonter automatiquement. L'information pluie est transmise au BCM.

La montée ou la descente complète de la vitre doit être réalisée en 4 secondes. Quel que soit le mode de montée/descente de la vitre, l'action s'arrête dès que la vitre arrive en butée. La détection de la butée se fait par la mesure du courant délivré par le moteur.

Le pilotage du moteur du lève vitre se fait par une commande PWM. Elle ne doit pas produire aucun bruit parasite.

Scénario dans le cas où l'ordre de lève vitre provient du BCM :

Le BCM transmet l'ordre de montée de la vitre de la portière et demande au DCM d'acquiescer l'opération. A la fin de l'opération lève vitre, le DCM transmet un acquiescement au BCM.

Si le DCM ne répond pas au bout de 5 s, le BCM renvoie à nouveau l'ordre de montée de la vitre. En cas d'absence de réponse, il renouvelle 4 fois la demande. Au bout de 5 demandes sans réponses, le BCM abandonne la demande d'acquiescement.

Description de la fonction affichage des messages

L'afficheur du tableau de bord doit afficher :

- l'état de verrouillage de la portière
- la vitesse du véhicule
- les erreurs de transmission/réception CAN du BCM (voir exigence sur la communication CAN)
- les pannes sur les switches de puissance de la portière

b. Exigences sur le BCM

La fréquence d'horloge du BCM est de 16 MHz. Cette fréquence est produite à partir d'un oscillateur à quartz à 4 MHz. La tolérance sur la fréquence du quartz est inférieure à 0.5 %.

c. Exigences sur la communication CAN

Installation matérielle du réseau CAN entre la portière et le BCM

Le module DCM communique avec le module BCM par un bus CAN. La longueur maximale du bus CAN entre le DCM et le BCM est de 5 mètres. La liaison CAN est assurée par une paire bifilaire. Le temps de propagation le long de la paire est de 5 ns/m. Le retard maximal introduit par un contrôleur CAN ou un transceiver CAN est estimé à 25 ns.

La connexion au bus CAN est assurée par un transceiver CAN de type NXP - TJA1041 (disponible sur les evaluation boards DEMO9S12XEP100).

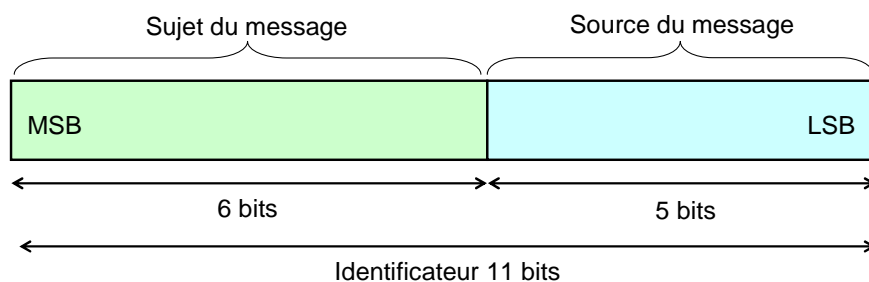
Exigences sur le débit binaire

Le bus CAN fonctionne en mode low speed. Le débit binaire brut doit être supérieur à 62.5 Kbits/s.

Exigences sur le format des trames CAN

Le Standard CAN 2.0A est utilisé. Seules des trames de données sont transmises. Les remote frame ne sont pas utilisées. Les données sont transmises par paquet de 8 octets.

Le choix des identifiants doit se conformer au format suivant :



Les messages provenant du BCM auront l'identificateur suivant : XXXXXX00000.

Gestion des erreurs de transmission et de réception sur le bus CAN

Dans le cas d'erreurs de transmission ou de réception, seules les entrées et les sorties de l'état Bus Off sont repérées. Pour le BCM, la sortie de l'état Bus Off se fait par une requête du contrôleur, après l'entrée dans l'état Bus Off. Après 5 entrées consécutives dans l'état Bus Off, une erreur est affichée sur le tableau de bord et la demande en cours est abandonnée.

Pour le DCM, la sortie de l'état Bus Off est automatique (selon la spécification du protocole CAN 2.0A : 128 occurrences de 11 bits récessifs).

Activité sur le bus CAN

Si le contrôleur CAN ne transmet rien pendant 2 secondes, celui-ci et le transceiver CAN passent en mode Listen Only.

Le contrôleur CAN et le transceiver CAN présentent plusieurs modes de fonctionnement (voir exigences sur la gestion de l'énergie). Le contrôleur CAN et le transceiver CAN doivent sortir des modes Sleep, Stand-By ou Power Down si un bit dominant circule sur le bus.

d. Exigences sur la gestion de l'énergie

Les modules DCM et BCM sont alimentés directement par la tension batterie (Vbat). L'application devra consommer le moins d'énergie possible.

Les entrées-sorties du microcontrôleur non utilisés seront configurées en entrée et fixés à un état logique '0'.

Modes de consommation des microcontrôleurs

Trois modes de consommation d'énergie sont définies pour les microcontrôleurs :

- Mode Run : fonctionnement normal.
- Mode Wait : si la période d'inactivité dépasse 10 secondes.
- Mode Stop : si le véhicule est arrêté et le moteur coupé.

Lors du passage en mode Stop, le BCM entre dans le mode Stop après s'être assuré que le DCM est entré en mode Stop. Avant d'entrer en mode Stop ou Wait, le DCM doit s'assurer que les switches de puissance sont entrés dans un mode de basse consommation.

Les microcontrôleurs sortent du mode Wait ou du mode Stop dans les cas suivants :

- Un appui sur un des boutons
- Une activité sur le bus CAN (bit dominant)
- La vitesse du véhicule dépasse 10 km/h
- Une détection de pluie
- Une commande par le RKE

Modes de consommation des contrôleurs CAN

Les contrôleurs CAN présentent 3 modes de consommation :

- Mode Normal : fonctionnement normal.
- Mode Sleep : fonctionnement basse consommation dans lequel le contrôleur CAN entre si aucune activité n'est détectée sur le bus CAN depuis 5 secondes.
- Mode Power Down : fonctionnement basse consommation dans lequel le contrôleur CAN entre si le microcontrôleur entre en mode Wait ou en mode Stop.

Le contrôleur CAN du DCM se réveille en cas de détection d'un état dominant sur le bus CAN.

Modes de consommation des transceivers CAN

Les transceivers CAN présentent 3 modes de consommation :

- Mode Normal : fonctionnement normal
- Mode Stand-By : fonctionnement basse consommation dans lequel le transceiver CAN entre si aucune activité n'est détectée sur le bus CAN depuis 5 secondes.

- Mode Sleep : fonctionnement basse consommation dans lequel l'interface CAN entre si le microcontrôleur entre en mode Wait ou en mode Stop.

L'entrée et la sortie du transceiver CAN des modes Stand-By et Sleep sont demandées par le contrôleur CAN.

e. Exigences en terme de diagnostic des fautes

Le module DCM vérifie l'état des switches de puissance et fait remonter toute défaillance au BCM, qui affiche un message d'alerte sur le tableau de bord.

Dès qu'un problème est signalé sur un switch, la commande du switch est coupée jusqu'à ce que le problème disparaisse. Lorsque le problème a disparu, le DCM transmet au BCM un message indiquant le recouvrement de la panne.

IV - Organisation et Planning

Un groupe de TP travaille ensemble au développement de l'application. Le groupe travaille à la spécification logicielle de l'application et à sa validation. Au sein de chaque groupe, le travail est divisé en binôme. Le travail de spécification, de codage et de test de chaque module de l'architecture logicielle est réparti entre les différents binômes. La répartition de ce travail est à définir par le groupe de TP.

Le tableau ci-dessous décrit le déroulement des séances. Les trois premières séances sont dédiées à la spécification logicielle de l'application et des différents modules. A l'issue de ces 3 séances, le groupe doit fournir un document de spécification logicielle et matérielle de l'application, chaque binôme fournit la spécification du ou des modules dont il a la charge.

Le travail de programmation ne pourra démarrer qu'une fois la spécification logicielle établi et transmis aux enseignants. Les séances 4 à 9 seront dédiées à la programmation et aux tests des modules et de l'application finale.

Durant la 6^e séance, une évaluation orale individuelle de 10 minutes sera menée. Chaque personne devra être capable d'expliquer le travail en cours et les choix matériels et logiciels.

A l'issue de la 9^e séance, les différents binômes rendent un rapport technique présentant le codage, les choix de configuration du microcontrôleur et les résultats de test de leur(s) module(s), ainsi qu'un code source compilable.

Séance	Description
1 – 3	Spécification de l'architecture logicielle et matérielle de l'application et des différents modules
Fin séance 3	Rapport de spécification logicielle et matérielle de l'application (par groupe) et des modules (par binômes)
4 – 9	Codage et tests des modules et des applications
Séance 6	Evaluation orale individuelle de 10 minutes
Fin séance 9	Rapport de codage et de tests des modules (par binôme)

Pour toute question technique portant sur les composants Freecale, envoyez les par e-mail aux encadrants de TP : alexandre.boyer@insa-toulouse.fr et patrick.tounsi@insa-toulouse.fr.

V - Notation

La notation portera sur les rapports techniques rédigés par groupe et par binôme, ainsi que sur l'évaluation orale. La répartition de la note est la suivante :

- 1/3 pour le travail de spécification
- 1/3 pour l'évaluation orale
- 1/3 pour le rapport de codage et de tests

Conseils pour les rapports de spécifications, de codage et de tests :

- Présenter le projet de manière synthétique et claire
- Utiliser des outils graphiques adaptés, facilitant la compréhension de vos algorithmes et le codage des applications
- Décrire l'architecture matérielle en faisant apparaître les branchements entre les différentes cartes, les entrées-sorties utilisées, les types de signaux, les fréquences, les niveaux de tension ...
- Décrire les solutions apportées pour respecter les contraintes fonctionnelles, de sécurité, de dimensionnement du bus CAN et de gestion de l'énergie
- Les codes sources doivent être suffisamment et correctement documentés. Dans le rapport de description du code source et dans le code source, des commentaires doivent être associés à chaque fonction et indiquer : le rôle de la fonction, les variables d'entrées et de sorties.
- Le format des rapports de tests sont libres. Ils doivent démontrer la validité des fonctions codées et le respect des contraintes définies dans le cahier des charges. Il est demandé de décrire les différents tests menés sur chaque fonction (vecteurs de test).

VI - Annexe 1 – Format des documents de spécifications

L'application que vous allez coder provient du cahier des charges du client. Avant de coder une application, il convient de traduire les spécifications du client en un algorithme et de s'assurer que cet algorithme répond aux spécifications du client.

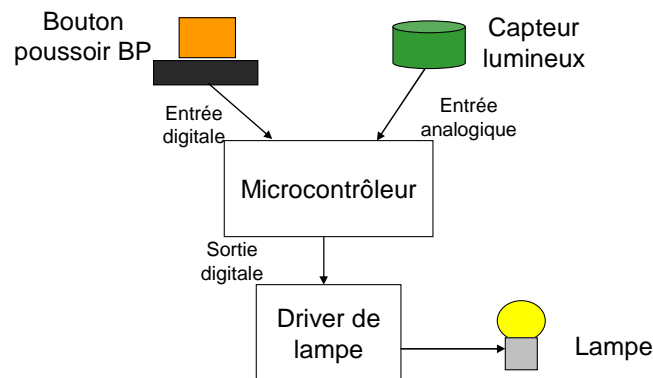
La spécification logicielle de l'application que vous allez coder doit se faire à 2 niveaux :

- la spécification de l'architecture logicielle, qui fait apparaître les différents modules composant l'application et leurs interactions
- la spécification de chaque module, qui détaille les fonctions de chaque module

Cette spécification vous permettra de définir votre algorithme. Pour réaliser cette spécification, différents formats peuvent être utilisés : texte, schéma-bloc, diagramme d'état, flowchart ...

Pour bien comprendre ces deux niveaux de spécifications et les différentes descriptions pouvant être employées, nous pouvons illustrer par l'exemple de spécification suivant :

« Un client souhaite réaliser une application d'allumage d'une lampe par l'appui sur un bouton et en fonction de la luminosité ambiante. La figure ci-dessous illustre l'architecture matérielle de l'application. La lampe est commandé par un driver de puissance, la commande est de type PWM, la fréquence de la PWM est de 100 Hz, le rapport cyclique de 50 %. La lampe s'allume lorsqu'on appuie sur le bouton poussoir ou si la tension analogique aux bornes du capteur lumineux est inférieure à V_{ref} . »



A partir de la spécification client, il est possible de décomposer l'application en plusieurs modules fonctionnels, que l'on peut décrire sous forme de texte (la fonction du module, les entrées-sorties, les contraintes ...) :

- Module « Détection_Appui_BP » : ce module détecte l'appui sur le bouton poussoir et transmet l'état du bouton poussoir.
- Module « Détection_obscurité » : ce module détecte si la luminosité ambiante passe sous le seuil ($V_{capteur} < V_{ref}$) et transmet l'état de lumineux ambiant.
- Module « Commande_lampe » : ce module décide de l'allumage de la lampe, en autorisant la commande PWM.
- Module « PWM_lampe » : ce module génère la commande PWM de la lampe à partir des paramètres de fréquence et de rapport cyclique, si le module Commande_lampe a donné son autorisation.

Chacun de ces modules peut aussi être décrit de manière un peu plus précise. La description d'une application ou d'un module sous forme de texte est certes générale, mais elle reste surtout adaptée à une description fonctionnelle. Elle n'est pas la plus adaptée à la description de l'architecture en module ou du séquençement des actions. D'autres outils sont plus adaptés :

Schéma-bloc ou schéma fonctionnel :

Il s'agit d'une représentation graphique d'un processus complexe présentant plusieurs unités. Le schéma fait apparaître les différents blocs du système, leurs entrées-sorties et les lignes d'action. La figure ci-dessous propose un schéma-bloc de l'application précédente et fait apparaître les 4 modules et leurs interactions. Ces différents modules pourront être traduits par une ou plusieurs fonctions.

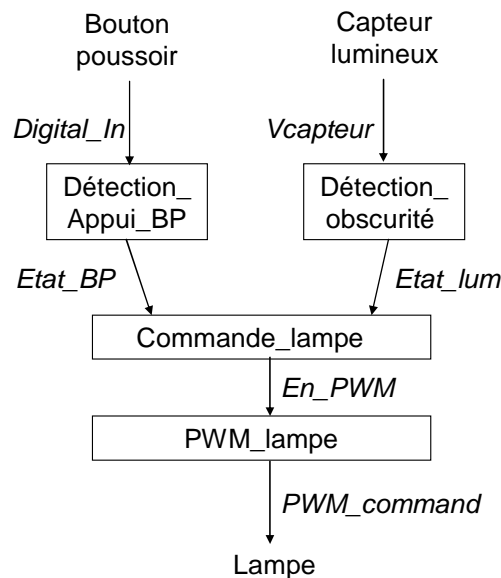
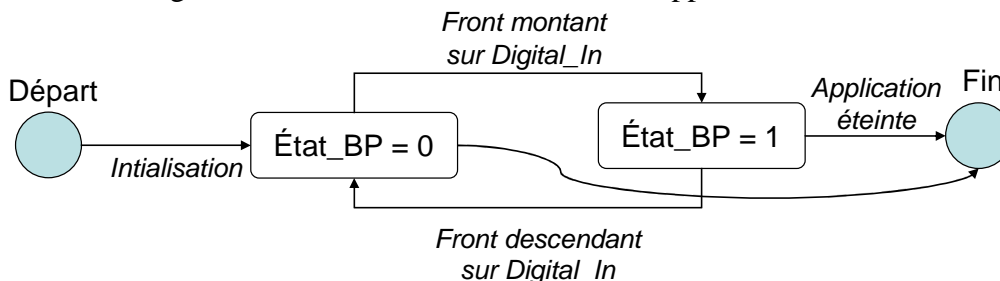


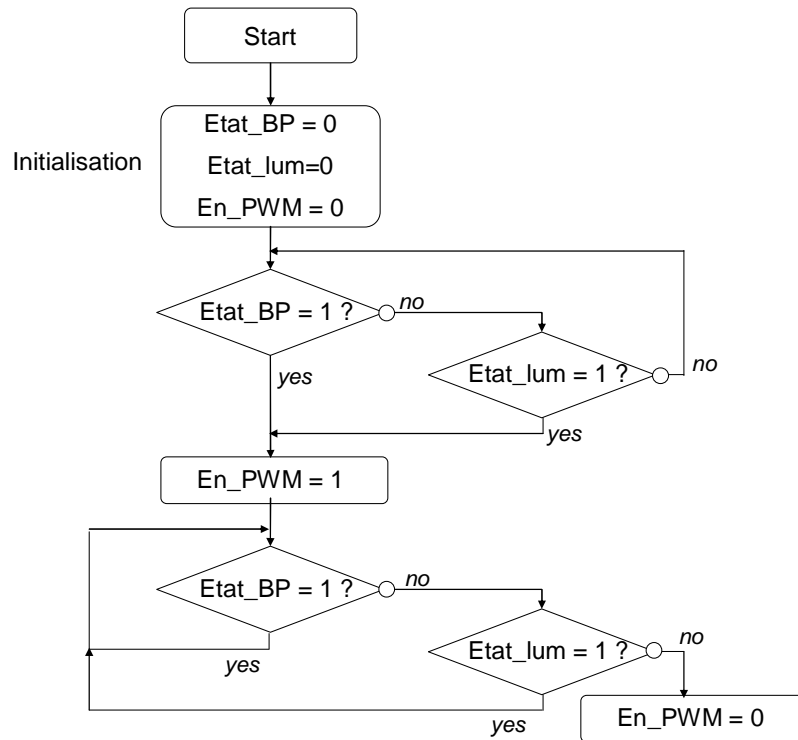
Diagramme d'état :

Le diagramme d'état permet de représenter les différents états pris par une machine à états finis et les conditions à remplir pour passer d'un état à un autre (transition). Un diagramme d'état présente l'avantage d'être directement traduisible sous la forme d'un algorithme. Par exemple, voici le diagramme d'état du module Détection_Appui_BP :



Flowchart ou algorithme :

Le flowchart permet de représenter graphiquement l'enchaînement des opérations. Comme le diagramme d'état, il est traduisible sous la forme d'un algorithme. La figure ci-dessous présente le flowchart de l'application :



Dans le cadre de ce BE, il n'y a pas d'obligation à utiliser ces différentes formes de description pour spécifier l'architecture logicielle que vous allez spécifier. Néanmoins, elles vous permettront de faciliter la création de vos algorithmes, la validation de vos spécifications par rapport à la spécification du client, le debug, la collaboration entre les différents groupes de travail.

VII - Annexe 2 – Présentation du matériel

Dans ce BE, nous disposons de plusieurs maquettes, de composants dédiés automobile fournis par la société Freescale Semiconductor, et de cartes d'interface. Les notes d'application des différents composants vous sont fournies.

L'ensemble des documents sont disponibles sur le site http://lesia.insa-toulouse.fr/~a_boyer.

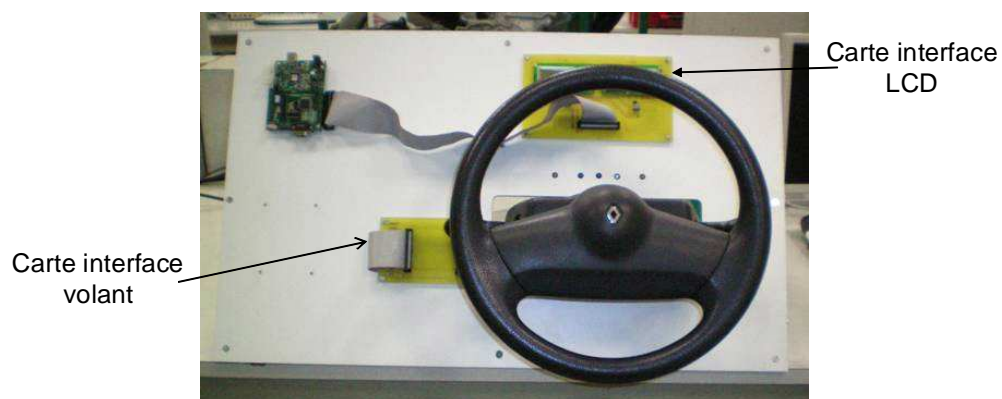
1. Les maquettes

Six maquettes dédiées à une application automobile sont proposées. Elles sont présentées ci-dessous accompagnées de leurs fiches signalétiques.

a. Module portière



b. Module volant



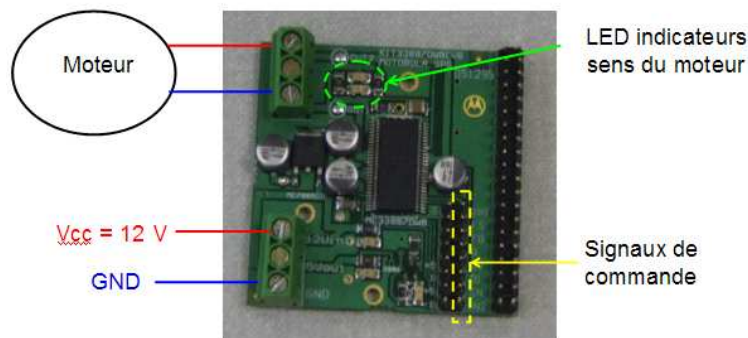
Le module volant inclut le volant et les commodos associés aux commandes du tableau de bord. Une carte d'interface volant est placée sur le module pour récupérer les commandes du tableau de bord. Il inclut aussi 2 afficheurs LCD montés sur une carte interface LCD, ainsi que plusieurs voyants lumineux.

2. Les cartes électroniques

Plusieurs cartes d'évaluation sont fournies par la société Freescale Semiconductor dans le cadre du BE automobile. L'interfaçage entre les différentes cartes sera assurée par une carte spécifique.

a. MC33887 – Pont en H

Ce composant intègre un pont en H et ses diodes de protection, ainsi que toute la partie logique d'interfaçage avec un contrôleur externe. Il propose en plus une sortie analogique renseignant sur le courant de sortie afin de mettre en place une boucle de courant, ainsi qu'une sortie logique Fault Status indiquant des conditions de fonctionnement anormales (sous tension, sur courant, surchauffe). Comme la plupart des composants automobiles, il possède plusieurs mode de fonctionnement permettant de réduire la consommation en énergie : mode normal ou mode sleep. Il est monté sur une carte de développement KIT33887DWBEVB, décrite sur la figure ci-dessous.



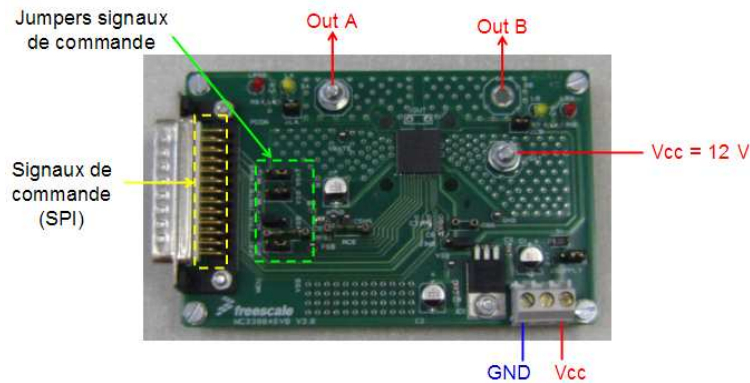
Fiche signalétique :

- Alimentation : 5 – 28 V (on travaillera sous 12 V)
- Courant DC max : 5 A (attention au cas du rotor bloqué)
- Fréquence max PWM : 10 KHz
- Signaux de commandes compatibles TTL/CMOS 5 V

Pour plus de détails, se reporter à la datasheet du composant et de la carte de développement KIT33887DWBEVB (http://lesia.insa-toulouse.fr/~a_boyer).

b. MC33984 – Dual High side switch

Ce composant intègre 2 commutateurs de puissance de type high side switchs et leurs diodes de protection, ainsi que toute la partie logique d'interfaçage avec un contrôleur externe. Ce composant peut être piloté à l'aide du protocole SPI ou directement en PWM. Ce composant peut faire passer un fort courant (jusqu'à 30 A en DC !). Il propose en plus une sortie analogique renseignant sur le courant de sortie permettant ainsi de mettre en place une boucle de courant, ainsi qu'une sortie logique Fault Status indiquant des conditions de fonctionnement anormales (sous tension, sur courant, surchauffe). Comme la plupart des composants automobiles, il possède plusieurs mode de fonctionnement permettant de réduire la consommation en énergie : mode normal ou mode sleep. Il est monté sur une carte de développement KIT33981BPNAEVB, présentée sur la figure ci-dessous.



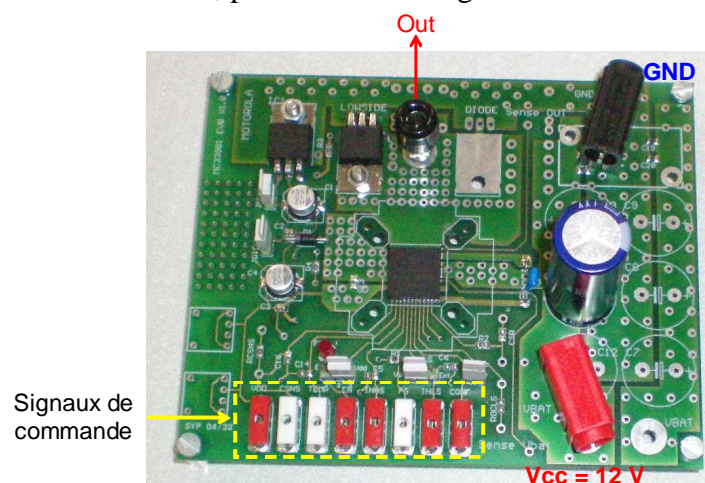
Fiche signalétique :

- Alimentation : 6 – 27 V (on travaillera sous 12 V)
- Courant DC max : 30 A (attention au cas du rotor bloqué)
- Fréquence max PWM : 300 Hz
- Signaux de commandes compatibles TTL/CMOS 5 V

Pour plus de détails, se reporter à la datasheet du composant et de la carte de développement KIT33984PNAEV3 (http://lesia.insa-toulouse.fr/~a_boyer).

c. MC33981 – Single High Side driver

Ce composant intègre un seul high side switch et ses diodes de protection, ainsi que toute la partie logique d'interfaçage avec un contrôleur externe. De plus, il peut piloter un N-MOSFET externe utilisé comme low side switch, permettant de réaliser un demi pont en H. Ce composant peut être commandé directement en PWM, jusqu'à une fréquence 60 KHz. Ce composant peut faire passer un fort courant (jusqu'à 40 A en DC !). Il propose en plus une sortie analogique renseignant sur le courant de sortie permettant ainsi de mettre en place une boucle de courant, ainsi qu'une sortie logique Fault Status indiquant des conditions de fonctionnement anormales (sous tension, sur courant, surchauffe). Comme la plupart des composants automobiles, il possède plusieurs mode de fonctionnement permettant de réduire la consommation en énergie : mode normal ou mode sleep. Il est monté sur une carte de développement KIT33984PNAEV3, présentée sur la figure ci-dessous.



Fiche signalétique :

- Alimentation : -16 – 41 V (on travaillera sous 12 V)
- Courant DC max : 40 A (attention au cas du rotor bloqué)
- Fréquence max PWM : 60 KHz

- Signaux de commandes compatibles TTL/CMOS 5 V

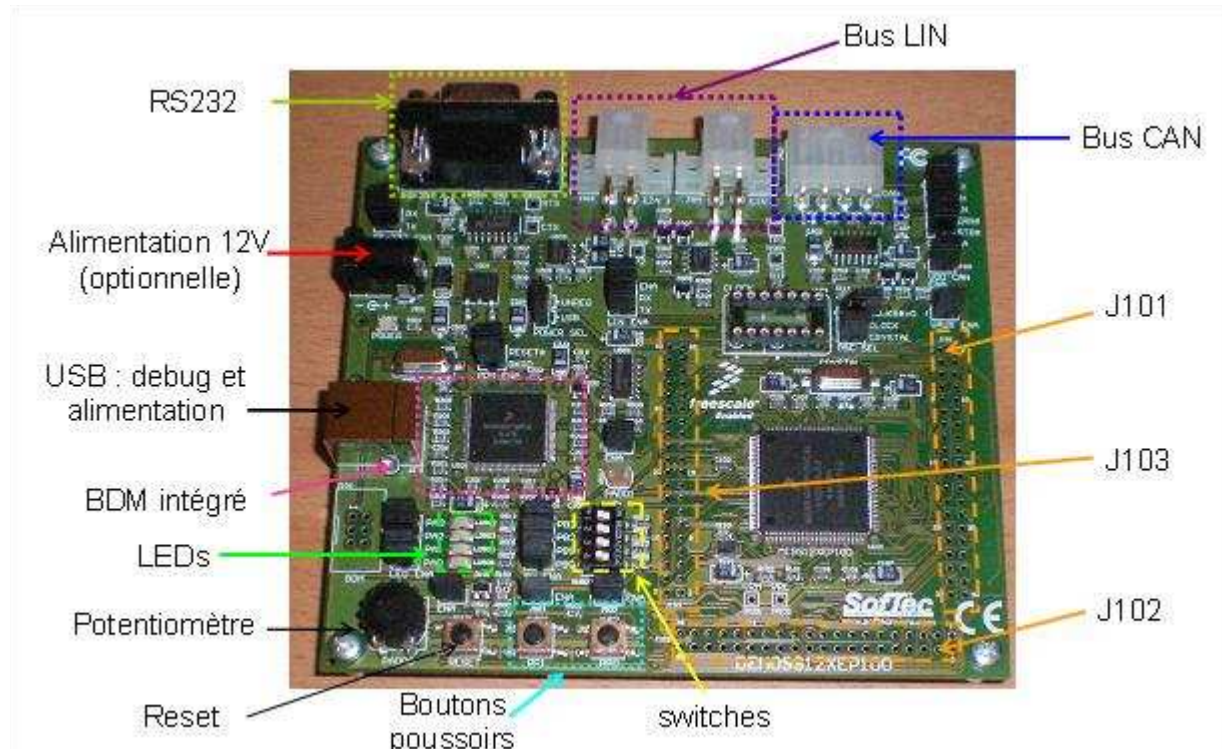
Pour plus de détails, se reporter à la datasheet du composant et de la carte de développement KIT33981PNAEVB (http://lesia.insa-toulouse.fr/~a_boyer).

d. Kit de développement DEMO9S12XEP100

Le kit de développement propose le microcontrôleur S12X dans sa version EP100 monté dans un boîtier QFP112. Cette carte relativement simple intègre le module de programmation/debug in-situ BDM. Il peut être alimenté soit par port USB, soit par un régulateur on-board à partir d'une alimentation DC à 12 V. Un quartz à 4 MHz fournit la référence d'horloge au composant. Sur cette carte, on trouve :

- 4 LEDs connectées aux pins PA0 à PA3
- 4 switches connectés aux pins PB0 à PB3
- 1 bouton poussoir de reset
- 2 boutons poussoirs connectés aux pins PP0 et PP1
- un potentiomètre connecté à l'entrée de conversion analogique numérique AN00
- une cellule photovoltaïque connectée à l'entrée de conversion analogique numérique AN01
- un connecteur (J301) pour un port série RS232 connecté à une interface SCI (PTS0/RXD0 et PTS1/TXD0), deux connecteurs (J303 et J304) connectés à des interface LIN (PTS2/RXD0, PTS3/TXD0 et PTS7/SS0), un connecteur (J403) connecté à une interface CAN (PTM0/RXCAN0 et PTM1/TXCAN0) (cf document demo9s12xep100_schematic.pdf)
- La plupart des pins du microcontrôleur sont disponibles à l'aide de 3 connecteurs 40 pins au pas de 1.27 mm (liste et placement des broches fournis dans le document demo9s12xep100_manual.pdf)

La figure ci-dessous présente la carte de développement DEMO9S12XEP100 utilisée dans ce BE.



40-Pin I/O Female Header Connector 1

1. VDD (5.0 V)
2. PE1
3. GND
4. RESET#
5. PS1
6. BKGD
7. PS0
8. PP7
9. PP0
10. PAD07
11. PP1
12. PAD06
13. PT0
14. PAD05
15. PT1
16. PAD04
17. PM4
18. PAD03
19. PM2
20. PAD02
21. PM5
22. PAD01
23. PM3
24. PAD00
25. PA7
26. PJ6
27. PA6
28. PJ7
29. PA5
30. PP2
31. PA4
32. PP3
33. PA3
34. PP4
35. PA2
36. PP5
37. PA1
38. PS2
39. PA0
40. PS3



40-Pin I/O Female Header Connector 2

1. PB0
2. PM0
3. PB1
4. PM1
5. PB2
6. PT2
7. PB3
8. PT3
9. PB4
10. PK0
11. PB5
12. PK1
13. PB6
14. PK2
15. PB7
16. PK3
17. PT4
18. PK4
19. PT5
20. PK5
21. PT6
22. PK7
23. PT7
24. PAD08
25. PE7
26. PAD09
27. PE6
28. PAD10
29. PE5
30. PAD11
31. PE4
32. PAD12
33. PE3
34. PAD13
35. PE2
36. PAD14
37. GND
38. PAD15
39. PE0
40. PP6



40-Pin I/O Female Header Connector 3

1. PS4
2. GND
3. PS5
4. GND
5. PS6
6. GND
7. PS7
8. GND
9. PM6
10. GND
11. PM7
12. GND
13. PH0
14. GND
15. PH1
16. GND
17. PH2
18. GND
19. PH3
20. GND
21. PH4
22. GND
23. PH5
24. GND
25. PH6
26. GND
27. PH7
28. GND
29. PJ0
30. GND
31. PJ1
32. GND
33. PJ6
34. GND
35. GND
36. GND
37. GND
38. GND
39. GND
40. GND



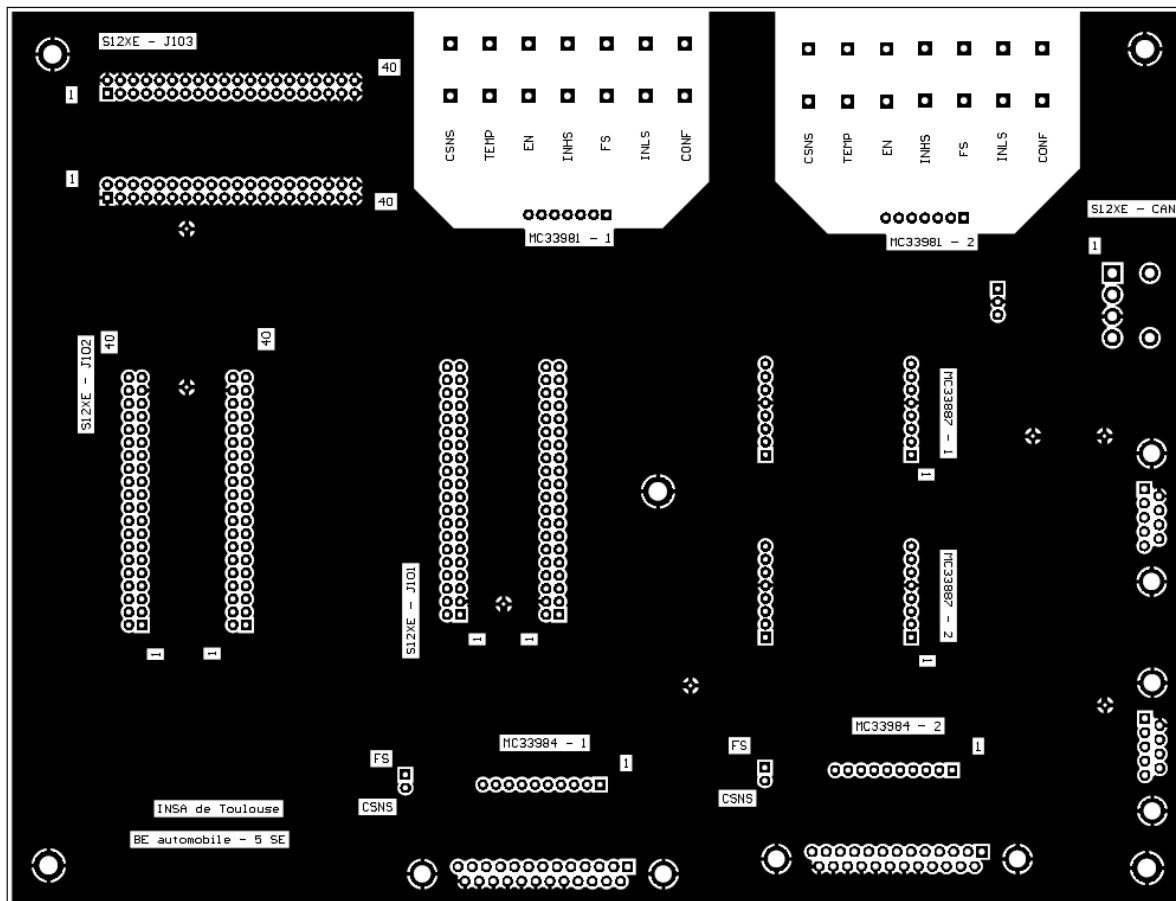
Pour plus de détails, se reporter à la datasheet du composant et de la carte de développement DEMO9S12XEP100 (http://lesia.insa-toulouse.fr/~a_boyer).

3. Carte d'interface

La carte DEMO9S12XEP100 représente l'élément central de chaque application. Afin de la connecter aux différentes cartes de puissance et aux autres cartes microcontrôleur par bus CAN, une carte d'interface a été développée. Celle-ci est présentée ci-dessous.

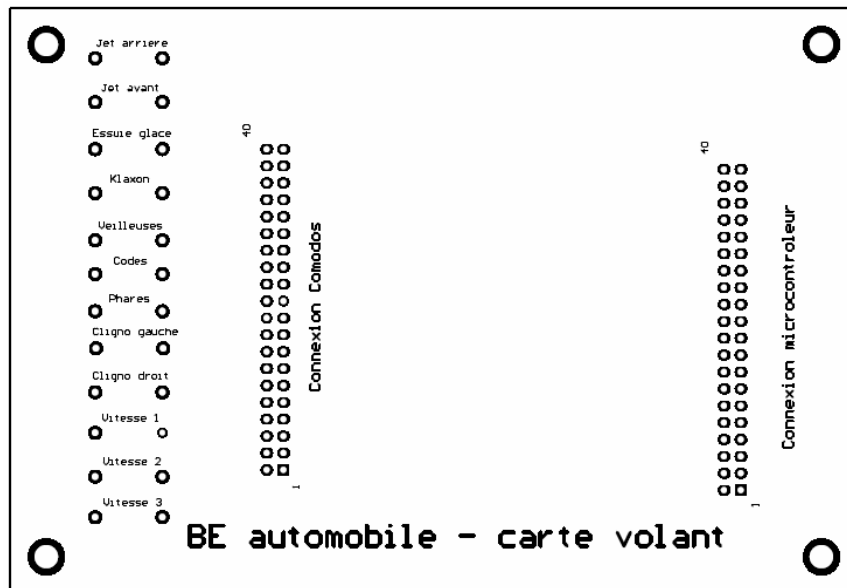
Les signaux provenant de la carte DEMO9S12XEP100 proviennent des 3 connecteurs 2*20 J101, J102, J103. Différents types de connecteurs sont utilisés pour se connecter aux cartes de puissance. En face de chaque broche des différents connecteurs de la carte sont placés des connecteurs tulipe. Ainsi, les interconnexions entre cartes peuvent être réalisées manuellement en plaçant des fils entre les connecteurs tulipes. L'assignation des broches du HCS12XE est donc laissée libre.

Les indications sur les connecteurs et le sens de connexion sont reportées sur les cartes.



4. Carte volant

Afin de récupérer les commandes des commodos, une carte a été montée sur le module volant. Un connecteur 2*20 permet de s'interfacer directement avec la carte DEMO9S12XEP100, en utilisant le connecteur J101.

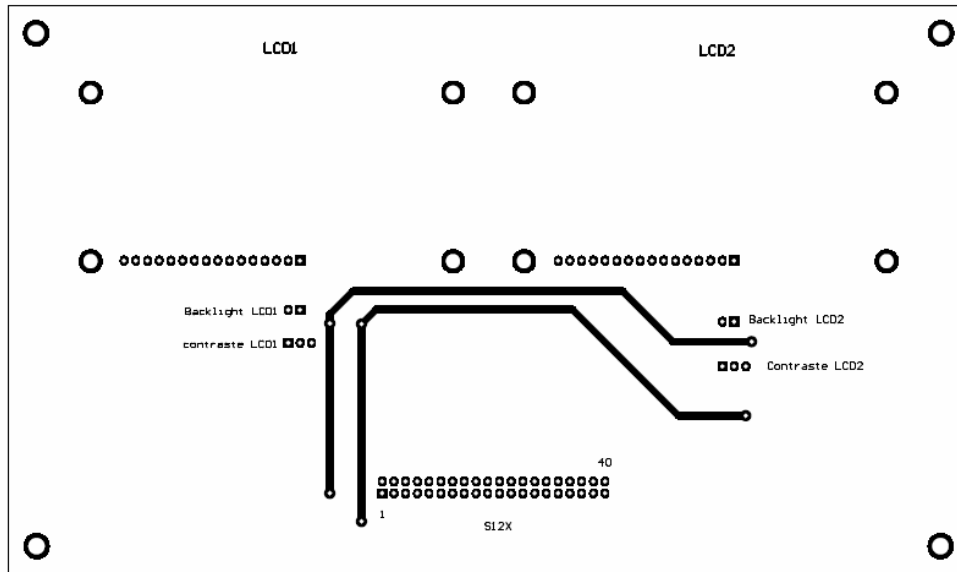


L'assignation des broches du S12X est la suivante :

Commandes	Broche S12X
Veilleuses	PP4
Codes	PA3
Phares	PP3
Clignotant gauche	PA4
Clignotant droit	PP2
Vitesse essuie glace 1	PA5
Vitesse essuie glace 2	PA6
Vitesse essuie glace 3	PA7
Klaxon	PA2
Essuie glace avant/arrière	PP5
Jet avant	PA1
Jet arrière	PA0
Direction volant	PAD00
Pédale d'accélération	PAD01

5. Carte afficheur LCD

Afin d'afficher des messages envoyés par le microcontrôleur central de l'application, deux afficheurs LCD sont montés sur le module volant. Ces 2 afficheurs peuvent être commandés par un microcontrôleur HCS12XE par l'intermédiaire de la carte décrite ci-dessus. L'interconnexion avec la carte DEMO9S12XEP100 se fait par le connecteur J101. Un potentiomètre et un jumper permettent de régler le contraste et allumer le rétroéclairage.



L'assignation des broches du connecteur J101 de la carte DEMO9S12XEP100 est la suivante :

Modules LCD		Broche S12X
LCD 1	RS	PP2
	R/W	PJ7
	Enable	PJ6
	DB0 → 7	PAD00 → 07
LCD 2	RS	PP5
	R/W	PP4
	Enable	PP3
	DB0 → 7	PA0 → 7

Référence de l'afficheur LCD : DisplayTech 162B (datasheet/Afficheur_LCD_162B.pdf)

VIII - Annexe 3 - Prise en main du matériel

1. *Prise en main de la carte MC33887*

Afin de comprendre le fonctionnement de cette carte, nous allons la faire fonctionner de manière manuelle. Pour cela, nous allons l'utiliser pour piloter une lampe que nous allons allumer. Celle-ci peut se piloter à partir d'un signal PWM. Pour ce premier essai, on pilotera la charge à l'aide d'une tension continue.

- Connecter la lampe sur la sortie du pont en H
- Connecter l'alimentation sans l'allumer (tension 12 V)
- A l'aide de la datasheet de la carte, mettre à '0' ou à '1' les différents signaux logiques de commande en plaçant ou non des jumpers sur les broches des signaux de contrôle
- Mettre sous tension, plusieurs LEDs sur la carte vous indiquent la mise sous tension du composant et du sens de fonctionnement du pont en H

2. *Prise en main de la carte MC33984*

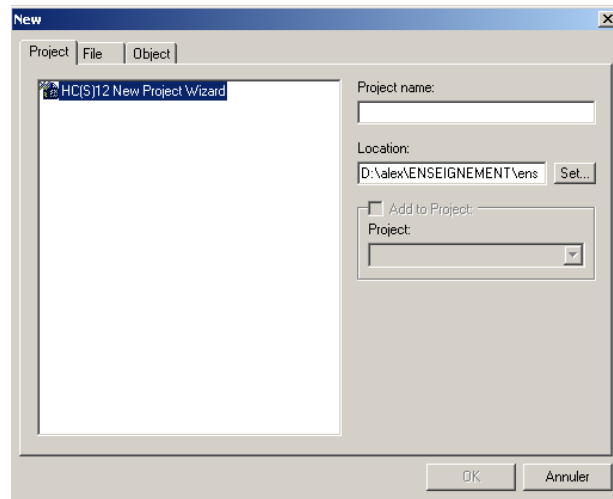
Il est possible de prendre en main cette carte par l'intermédiaire du logiciel SPIGEN. Cependant, nous préférons utiliser une approche manuelle, sachant qu'il est possible de fixer l'état des différents signaux logiques directement sur la carte. La charge utilisée est une lampe.

- Connecter la lampe sur une des sorties (SA ou SB)
- Connecter l'alimentation sans l'allumer (tension 12 V)
- A l'aide de la datasheet de la carte, mettre à '0' ou à '1' les différents signaux logiques de commande. Pour cela, placer correctement les jumpers présents sur la carte
- Mettre sous tension, plusieurs LEDs sur la carte vous indiquent la mise sous tension du composant et l'état de la sortie

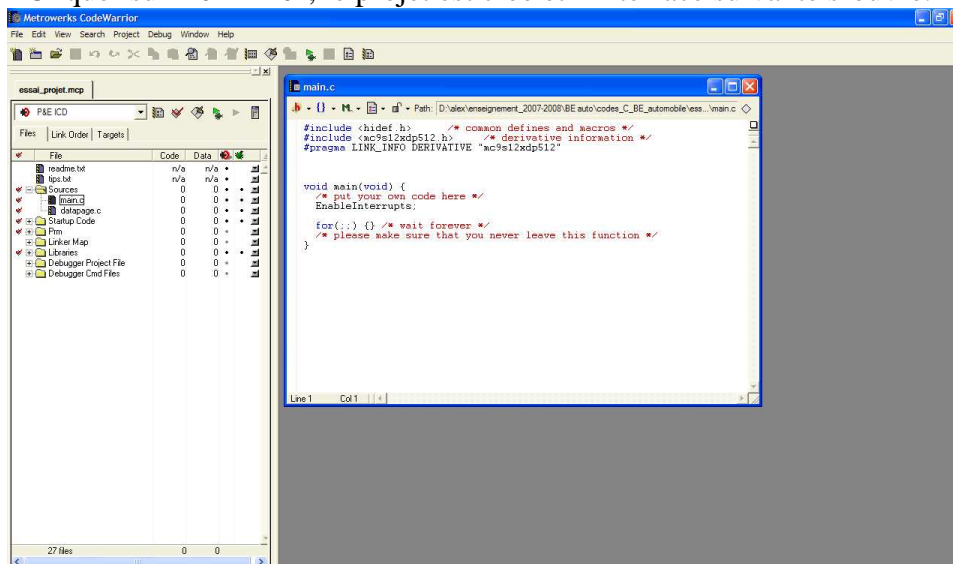
3. *Prise en main de la carte de développement DEMO9S12XEP100 et de l'outil de programmation Freescale CodeWarrior v4.7*

L'outil Freescale CodeWarrior v4.7 permet de développer les fichiers exécutables qui seront implantés dans le microcontrôleur MC9S12XEP100, mais aussi de réaliser du debug in-situ et d'implanter les programmes dans la mémoire flash du composant. Nous allons commencer par décrire comment construire une application à l'aide de Code Warrior.

- Lancer Freescale CodeWarrior v4.7
- Construire un nouveau projet. Pour cela, cliquer **File>New** et sélectionner **HC(S)12 New Project Wizard**. Nommer votre projet, spécifier le chemin d'accès des fichiers puis cliquer **OK**.



- Le project wizard est lancé et vous aide à construire votre projet en plaçant automatiquement les fichiers utiles (bibliothèques, fichiers de lien, fichiers header, ...) dans votre répertoire. Pour cela, différents choix vous sont proposés.
- D'abord, vous devez sélectionner le composant : **MC9S12XEP100**. Ensuite, il vous est demandé si vous souhaitez utiliser la XGATE. Il s'agit d'un coprocesseur permettant de gérer les interruptions et réduire la charge du microcontrôleur. Pour nos applications, nous ne l'utiliserons pas, sélectionnez **Single Core (HCS12X)**.
- Choisissez ensuite le langage de programmation. On programmera en langage C.
- L'outil PC-lint ne sera pas utilisé. Le startup code sera en C ANSI : **ANSI startup code**. Aucun format en point flottant ne sera utilisé. On sélectionne une mémoire de type **Banked Memory**.
- Enfin, on sélectionne l'outil **Softec HCS12** pour programmer le microcontrôleur et réaliser le debug in-situ.
- Cliquer sur **Terminer**, le projet est créé et l'interface suivante s'ouvre.



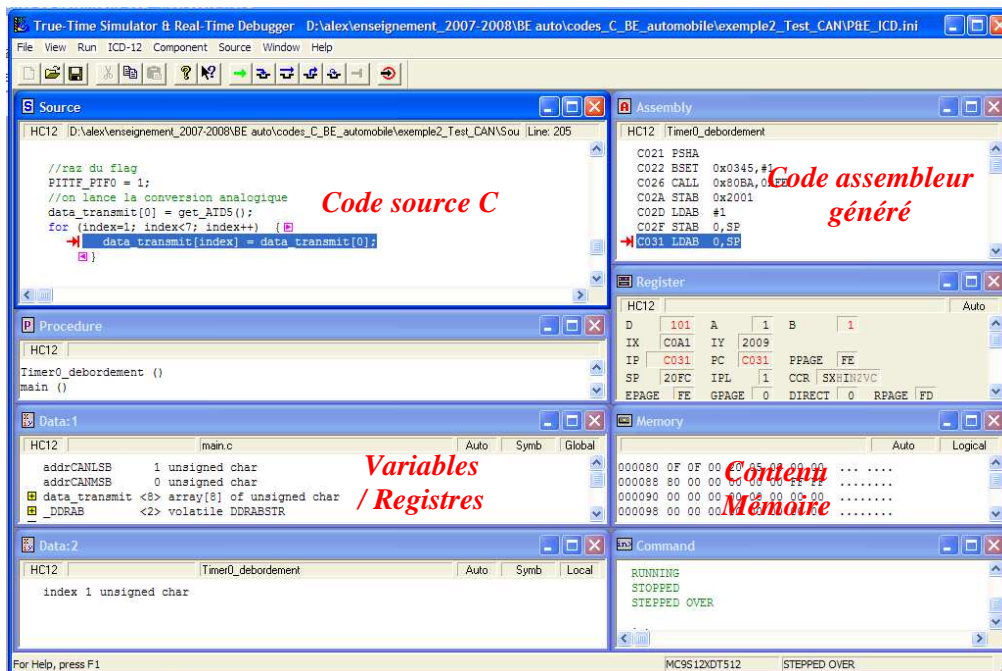
- Vous n'avez plus qu'à écrire le code source. Un exemple de code source vous est donné dans les fichiers sources.

Une fois votre code source écrit, il vous faut le compiler et générer le code exécutable qui sera implanté à l'intérieur du microcontrôleur. Pour cela, on utilise les boutons ci-dessous (**Compile, Make, Debug**).



La commande Debug permet de lancer le mode Debug et d'implanter le code sur le composant. Pour lancer le debug in-situ, il faut sélectionner la cible **Softec HCS12**, puis cliquer sur **Debug**.

La fenêtre du debugger Hi-Wave s'ouvre, après vous avoir demandé l'autorisation de réécrire la mémoire programme. La figure ci-dessous présente les différents volets du simulateur. La barre d'outil suivante permet de réaliser différents modes d'exécution (run, pas à pas). Plusieurs points d'arrêt peuvent être inclus par un clic droit dans la fenêtre Source et sélection de Set Breakpoint.



Un exemple de code source vous est fourni pour vous aider à démarrer (**notes application/exemple_code_source_S12XE.c**). Copiez le sur votre PC. Pour charger ce programme dans la mémoire flash du microcontrôleur, vous devez l'inclure à un projet. A partir d'un projet construit à partir du Project Wizard, vous pouvez :

- soit copier le contenu du fichier exemple_code_source_S12X.c dans le fichier main.c
- soit supprimer le fichier main.c actuel, renommer le fichier exemple_code_source_S12X.c en main.c et l'ajouter dans le fichier source du projet en faisant un clic droit au dessus de l'explorateur projet, puis en cliquant sur Add Files.

Il ne reste plus qu'à sauvegarder ce fichier, compiler le code et le charger en mémoire. Le code devrait s'exécuter correctement sur le HCS12XE.