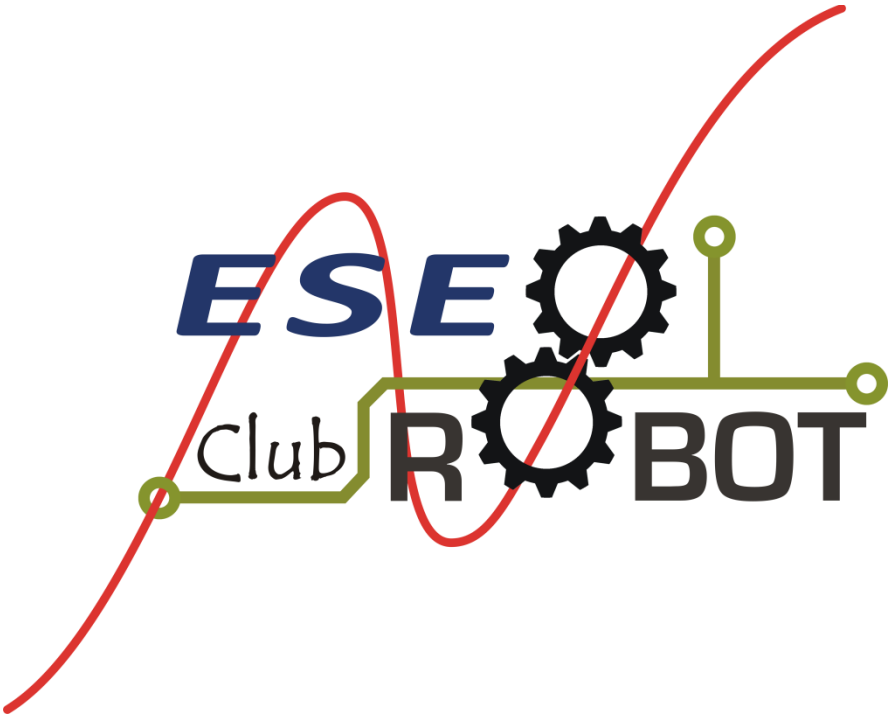
CAHIER DE QUALITE LOGICIELLE



30/05/2010

Normes de développement logiciel

Ce document a pour objectif de fournir des normes et des conseils sur la programmation au club robotique de l'ESEO

Cahier de qualité logicielle

NORMES DE DEVELOPPEMENT LOGICIEL

ACRONYMES

Les acronymes suivants seront utilisés dans le reste du document :

MISRA: Motor Industry Software Reliability Association

RQS : responsable de la qualité software.

NORMES DE CODAGE

Les règles suivantes sont pour la plupart issues des règles MISRA, qui sont très souvent employées dans les systèmes embarqués.

Environnement

Tout code embarqué dans le robot devra être écrit en ISO-C99 (ISO/CEI 9899:1999) sans extensions. Toute exception devra être validée par le RQS.

Set de caractères

Les caractères utilisés doivent être conformes avec l'ISO-8859-15.

Les trigraphes ne doivent pas être utilisés (Les trigraphes sont des séquences de trois caractères commençant par deux points d'interrogations. Ils permettent de remplacer les caractères qui ne sont pas accessibles sur tous les environnements.).

Les caractères codés sur plusieurs octets ne doivent pas être utilisés.

Commentaires

Les commentaires ne doivent pas être imbriqués

Les commentaires ne doivent pas contenir de sections de codes. Sauf pour les tests !

Identificateurs

Les identificateurs doivent être d'une longueur de moins de 31 caractères.

Des identificateurs de portée différente ne doivent pas porter le même nom. Les identificateurs de membres de structures font exception à cette règle

(Ex : Si une variable global et une variable déclarée dans une fonction, il peut y avoir confusion par le programmeur).

Types

Les types de base *char*, *int*, *short*, *long*, *double* et *float* ne doivent pas être utilisés. Utilisez à la place les types définis par « QS_types.h » (Exception faite au code de la carte propulsion !).

La représentation binaire d'un type flottant ne doit jamais être exploitée par le programmeur.

Les noms utilisés dans des *typedef* ne doivent pas être réutilisés.

Constantes

Les valeurs en octal ne doivent pas être utilisées. L'utilisation de 0 est acceptée. En effet, en langage C pour utiliser l'octale, il suffit de mettre un « 0 » devant un nombre/chiffre.

Déclarations et définitions

Tout objet ou fonction doit être déclaré avant toute référence. On doit instancier l'objet, c'est-à-dire réserver de l'espace mémoire avant de fournir l'adresse de l'objet.

Les identificateurs locaux ne doivent pas masquer des identificateurs de plus grande portée.

Toute déclaration d'objet doit être faite avec une portée limitée à la fonction si possible.

Toute définition d'objet avec pour portée le fichier doit être *static (placé en tête des fichiers)*.

Toute autre définition d'objet doit être faite dans Global_vars, comme indiqué dans la section « Utilisation du Package Qualité Soft ».

Initialisations

Toutes les variables doivent être initialisées avant d'être utilisées.

L'initialisation d'une structure ou d'un tableau doit se faire avec des accolades dont le contenu correspond à la déclaration du type de la structure ou du tableau .

(Ex :

```
typedef const struct
{
      Uint8 nb_elements;
      element_t elements[];
}element_list_t;
```

La structure précédente peut-être remplie de la façon suivante :

```
element_list_t color_list = {2,{"Bleu","Jaune"}};
```

)

Les énumérations doivent, au choix, ne pas avoir de valeurs numériques associées aux valeurs de l'énumération, n'avoir de valeur associée qu'à la première valeur de l'énumération, ou avoir une valeur associée à chaque valeur de l'énumération.

Opérateurs

Les opérandes de droite des opérateurs && ou || ne doivent pas avoir d'effets de bord.

(Ex : La condition suivante possède un effet de bord sur l'opérande de droite: `if(x && y++)`. Pour l'opérande de gauche, les effets de bord sont autorisés à condition de bien connaître la différence entre post-fixé et préfixé).

L'opérateur d'assignement ne doit pas être utilisé dans des expressions renvoyant des valeurs binaires (Jamais de '=' dans une condition !).

Les opérateurs bit-à-bit ne doivent pas être mélangés avec les opérateurs logiques. (On ne mélange pas &&, || et ! avec &, | et ^).

Les opérations bit-à-bit ne doivent pas être utilisées sur des variables signées.

L'opérande droite d'un opérateur de décalage doit avoir une valeur comprise entre 0 et la taille en bits de l'opérande gauche moins 1.

L'opérateur moins unaire ne doit pas être utilisé sur une expression non signée.

L'opérateur *sizeof* ne doit pas être utilisé sur des expressions ayant des effets de bords.

L'opérateur virgule ne doit pas être utilisé hors du contrôle d'une boucle *for*.

Conversions

Les conversions implicites provoquant des pertes de données ne doivent pas être utilisées.

Les conversions explicites redondantes ne doivent pas être utilisées.

Les conversions de ou vers un type pointeur sont proscrites.

Expressions

La valeur d'une expression doit être la même dans tout ordre d'évaluation que la norme autorise.

L'arithmétique sur des valeurs de précisions différentes doit utiliser des conversions explicites pour générer le résultat attendu.

Les expressions en virgule flottante ne doivent pas être testées en égalité ou en inégalité stricte.

Contrôle du flux d'exécution

Il ne doit pas y avoir de code mort.

Toute instruction non nulle doit avoir un effet de bord.

Une instruction nulle doit être seule sur sa ligne.

Les labels ne doivent pas être utilisés hors des *switch*.

L'instruction goto est interdite.

Toute instruction switch doit comporter un *default*.

Toute instruction switch doit avoir au moins un *case*.

Les variables en virgule flottante ne doivent pas être utilisées comme compteur de boucle.

Seules les expressions servant au contrôle d'une boucle doivent apparaitre dans le contrôle d'une boucle *for*.

Fonctions

Les fonctions doivent toujours être déclarées avec le fichier pour portée.

Les fonctions avec un nombre variable d'arguments ne doivent pas être utilisées.

Les fonctions doivent avoir un prototype unique visible à la définition et à chaque appel.

Les fonctions ne doivent pas s'appeler elles même, directement ou indirectement. Toute exception doit être validée par le RQS (Le récursif est difficile à maitriser).

Tout appel de fonction doit être conforme au prototype de la fonction.

Toutes les fonctions doivent avoir un type de retour explicite.

La valeur de retour d'une fonction renvoyant *void* ne doit pas être utilisée.

Les expressions de type *void* ne doivent pas être passées en paramètres de fonctions.

Il doit exister une instruction return pour chaque branche de sortie d'une fonction.

Pour les fonctions renvoyant *void*, l'instruction return ne doit pas comporter d'expression.

Directives préprocesseur

Les directives *include* ne doivent être précédées que de directives préprocesseur et de commentaires.

#undef ne doit pas être utilisé. Toute exception doit être validée par le RQS (Exception au code balise avec les LEDS).

Une macro comportant des arguments doit être appelée avec tous ses arguments.

Toute utilisation de *#pragma* doit être documentée, expliquée, et validée par le RQS (Cette expression est une directive de compilation spécifique liée au microprocesseur).

Pointeurs et tableaux

Aucune opération arithmétique ne doit être effectuée sur un pointeur.

Un pointeur sur fonction ne doit pointer que sur un type de fonction.

(Exemple d'un pointeur sur une fonction :

```
void (*p)(Uint8, Uint8) ;
```

On déclare donc un pointeur sur une fonction renvoyant void et qui prend en argument deus Uint8)

Le pointeur *NULL* ne doit pas être déréférencé (Le pointeur NULL est une adresse fictive ne correspondant à aucune zone de mémoire accessible).

Structures et unions

Les types unions ne doivent pas être utilisés. Toute exception doit être validée par le RQS.

(Alors que les structures permettent la concaténation d'informations, les unions visent en quelque sorte à en permettre la superposition. Une union permet ainsi d'interpréter de différente manière un même emplacement en mémoire. Elles étaient très utilisées lorsque les machines possédaient très peu de mémoire).

Tous les membres d'une structure doivent être nommés et tout accès doit se faire par leur nom.

Librairies standards

Les mots réservés et les noms de fonctions des librairies standards ne doivent pas être redéfinis ou réutilisés.

L'allocation dynamique ne doit pas être utilisée. Toute exception doit être validée par le RQS.

L'indicateur d'erreur *errno* ne doit pas être utilisé.

La macro *offsetof* dans la librairie <*stddef.h*> ne doit pas être utilisée.

<*local.h*> et la fonction *setlocale* ne doivent pas être utilisées.

Setjmp et *longjmp* ne doivent pas être utilisés (Permet des sauts dans les programmes comme goto).

Les fonctions de gestion du temps de la librairie <*time.h*> ne doivent pas être utilisées (Sur un microprocesseur le temps est défini par les timers …).

CONVENTIONS DE NOMMAGE ET DE PRESENTATION

En-tête des fichiers

Tout fichier de code du robot doit commencer par l'en-tête suivant :

```
/*
 *  Club Robot ESEO 2006 - 2007
 *  Game Hoover
 *
 *  Fichier : SuperCAN.h
 *  Package : SuperVision
 *  Description : fonctions de gestion du périphérique CAN
 *  Auteur : Kim, modifié par Jacen (2006-2010)
 *  Version 20070211
 */
```

Les années indiquées doivent couvrir de l'écriture du code à sa dernière utilisation, elles sont suivies des noms des robots ayant embarqué ce code.
Le nom du fichier doit à lui seul laisser présumer de son contenu.
Le package est le nom de l'unité du robot responsable de ce segment de code.
La description doit être aussi précise et synthétique que possible, sans ambiguïté.
Le nom d'auteur peut être un pseudonyme, mais il doit dans tous les cas permettre de retrouver l'auteur du code facilement. En cas de modification, le nom de la personne modifiant le code doit être mentionné, avec ses années de présence au club.
Le numéro de version doit être tenu à jours même en cas de modification mineure du code ou de ses commentaires. Il correspond à la date de la dernière modification au format AAAAMMJJ.

Indentation

L'indentation doit se faire avec des tabulations. Le style d'indentation recommandé est le style BSD/Allman, (cf http://fr.wikipedia.org/wiki/Style_d'indentation), il est cependant toléré de placer les accolades ouvrantes en fin de ligne, tant pour les fonctions que pour les structures conditionnelles.

Commentaires

Les commentaires seront :

- nombreux.

- utiles.
- utilisés à bon escient.
- En français correct.
- Explicites pour un étranger à l'art (dans la mesure du possible).

Langue d'écriture du code

Le code devra être rédigé en une seule langue, idéalement l'anglais, pour tout ce qui concerne le nommage des variables, des fonctions, des types et des fichiers, ceci pour éviter les confusions entre variables de même nom, en anglais et en français, notamment.

Conventions de nommage

Pour une plus grande lisibilité du code les conventions de nommage suivantes devront être respectées :

- Le nom des types définis par des énumérations s'achèvera par _e, comme bool_e.
- Le nom des autres types définis s'achèvera par _t, comme time_t.
- Le nom des alias (#define) devra être écrit en majuscules, les mots seront séparés par des underscore, et ne devront pas finir par l'extension d'un type de fichier courant (c, h, cpp, lkr, gld, coff, hex). Par exemple LONGUEUR_TERRAIN, ou AREA_LENGTH si vous avez choisi de coder en anglais.
- Les fonctions seront nommées en minuscules, les séparations des mots composant le nom seront marqués par des underscores '_'. Les fonctions liés à un module particulier (comme le bus CAN, les PWM, un CPLD) auront un nom commençant par le nom du module en majuscules suivi d'un nom explicite (par exemple PWM_init).

UTILISATION DU PACKAGE QUALITE SOFT

Le Package Qualité Soft ou PaQS est une bibliothèque de fonctions écrite fin 2008 pour fournir une API simple à utiliser, regroupant toutes les fonctionnalités matérielles attendues pour le Club Robotique de l'ESEO, permettant d'uniformiser les programmes fonctionnant dans le robot et de faciliter leur portage du dsPIC30F6010A vers une autre architecture si ce changement venait à l'ordre du jour.

Le PaQS a été écrit avec plusieurs objectifs :

- Fournir le code de configuration du microprocesseur
- Fournir le code d'interfaçage avec le matériel
- Restreindre l'utilisation du matériel aux seules fonctions utiles aux développeurs
- Gérer un maximum d'IT à la place des développeurs
- Exécuter en tâche de fond tout ce qui peut l'être
- Minimiser son empreinte sur les performances du microcontrôleur

Il est constitué des fichiers suivants :

- Librairie de notre microprocesseur :
  « libp30F6010A-coff.a »

- Linker de notre microprocesseur :
  « p30f6010A.gld »

- Licence du package (CeCILL-C) :
  « LICENCE.txt »

- Les fichiers sources (.c) :
  QS_adc.c, QS_all.c, QS_ax12.c, QS_can.c, QS_can_over_uart.c, QS_configBits.c, QS_DCMotor.c, QS_extern_it.c , QS_global_vars.c, QS_ports.c, QS_pwm.c, QS_qei.c, QS_qei_on_it.c, QS_servo.c, QS_Step_motor.c, QS_timer.c, QS_uart.c, QS_watchdog.c

- Les headers (.h) :
  p30f6010A.h, QS_adc.h, QS_all.h, QS_ax12.h, QS_can.h, QS_CANmsgDoc.h, QS_CANmsgList.h, QS_can_over_uart.h, QS_configCheck.h, QS_config_doc.h, QS_DCMotor.h, QS_extern_it.h, QS_global_vars.h, QS_macro.h, QS_ports.h, QS_pwm.h, QS_qei.h, QS_qei_on_it.h, QS_servo.h, QS_Step_motor.h, QS_timer.h, QS_types.h, QS_uart.h, QS_watchdog.h

Le PaQS nous permet une utilisation haut-niveau de notre microprocesseur. A chaque projet utilisant le dsPIC30F6010A, on place le dossier QS à coté des fichiers sources de notre projet. Grâce à ce dernier le microprocesseur est facilement configurable et programmable. Dans le code, lors de l'utilisation d'un module, il suffit d'inclure le module QS correspondant (Ex: QS_timers.h).