projet
document

Louison GOUY Yiying WEI

November 26, 2021

name

abstract

It's not you can use C to generate good code for hardware. If you think like a comupter writing C actually makes sens.

Linus Torvalds

Table des Matières

Liste des figures

1 Le langage C

Le langage C est un langage combiné, il a les caractéristiques des langages évolués (boucles itératives etc.) associé à des fonctionnalités des langages assemblés (décalage de bit, adressage indirect généralisé etc.). C'est la combinaison de ces deux caractéristiques qui font la force du langage [2]. Sa proximité avec l'assembleur le rendant très efficace, il est ainsi devenu le langage indispensable dans la programmation des applications comme l'automatique, la robotique, les OS ect. Cette même proximité impose peu de contraintes à l'utilisateur sur la structure de son programme. Aussi, il est possible d'écrire des fonctions avec plusieurs points de sorties, ou encore, d'échapper à une boucle avant son terme. Là où certains trouveront une grande souplesse, les critiques le considérerons trop permissive. On notera qu'un certain nombre d'organismes officiels proposent un ensemble de règles visant, tout en conservant son efficacité, à éviter les problèmes liés à une programmation peu soignée. L'Agence National de la Sécurité des Systèmes Informatiques française (ANSSI) propose un rapport complet, *Règles de programmation pour le développement sécurisé de logiciels en langage C* [1], visant à "favoriser la production de logiciels C plus sécurisés, plus sûrs, d'une plus grande robustesse et portables". Il sera la référence durant ce projet.

2 Programme de base

Dans un premier temps un nouveau projet est créé de type "GCC C ASF Board project". Microchip studio génère alors une arborescence de fichiers dont un main.c. Ce dernier est étudié de manière globale puis affinée par étape dans la section suivante.

2.1 Vue globale

Cette partie détaille le fonctionnement du programme de base.

```
1  #include <asf.h>
2
3  int main (void)
4  {
5    system_init();
6
7    /* Insert application code here, after the board has been initialized. */
8
9    /* This skeleton code simply sets the LED to the state of the button. */
10   while (1) {
11     /* Is button pressed? */
12     if (port_pin_get_input_level(BUTTON_0_PIN) == BUTTON_0_ACTIVE) {
13       /* Yes, so turn LED on. */
14       port_pin_set_output_level(LED_0_PIN, LED_0_ACTIVE);
15     } else {
16       /* No, so turn LED off. */
17       port_pin_set_output_level(LED_0_PIN, !LED_0_ACTIVE);
18     }
19   }
20 }
```

La première ligne permet d'inclure la bibliothèque asf et ainsi de profiter du niveau d'abstraction mis à disposition par Microship. La suivante, bien connu des développeurs C, est le point d'entré du programme. C'est la première fonction exécutée. La ligne 5 `system_init();` a été générée automatiquement par le logiciel à la création du projet. C'est elle qui nous offre ce niveau d'abstraction en initialisant les horloges et les entrées/sorties. Elle est spécifique à la cible utilisée, dans notre cas la carte Microchip SAMD21 Xplained Pro. La ligne 10 correspond à l'implémentation d'une boucle infinie. Cette dernière permet de lire l'état du bouton (ligne 12) à chaque exécution de la boucle. La condition suivante d'éclanche alors le résultat souhaité l'allumage ou l'extinction de la LED0.

2.2 Affinage

Include ASF
L'Advanced Software Framework (ASF) fournit un riche ensemble de pilotes éprouvés et de modules de code développés par des experts pour réduire le temps de conception. Il simplifie l'utilisation des microcontrôleurs en fournissant une abstraction au matériel par le biais de pilotes et de middlewares à forte valeur ajoutée. ASF est une bibliothèque de code gratuite et open-source conçue pour être utilisée lors des phases d'évaluation, de prototypage, de conception et de production.

System_init
Au début du `main` la fonction `system_init()` est appelée. Comme son nom l'indique elle a pour but d'initialiser le système. Elle est définie dans le fichier `system.c` et consiste en un simple appel successif à cinq fonctions de configuration : `system_clock_init(); system_board_init();`

`_system_events_init();` `_system_extint_init();` et `_system_divas_init();`. Elles jouent chacune un rôle essentiel dans l'initialisation de carte comme l'horloge ou les entrées sorties par exemple.

Boucle infinie

Implémenté à travers un `tant que VRAI`, cette ligne n'est pas difficile à comprendre mais il peut être intéressant d'en établir le contexte. Le guide des bonnes de pratiques de l'ANSSI indique toutefois que la forme d'une boucle infinie est bien `while(1)` et non `for(;;)`

De manière générale, le bouclage répète un jeu d'instruction jusqu'à se qu'une condition particulaire soit atteinte. On définit une boucle infinie dès lors que cette condition n'arrive jamais en raison d'une caractéristique inhérente à la boucle. Dans notre cas la condition de sortie serait VRAI=FAUX. C'est impossible !

Du point de vue matériel l'utilisation d'une boucle infinie permet de borner le programme compteur (PC) dans un espace mémoire bien définit. Le compilateur devrait l'interpréter par un jump ou jmp. Le mieux est probablement de le vérifier. Un fichier `loop.c` est créé, volontairement le plus simple possible.

```
1 /* file loop.c */
2 void main(void){while(1);}
```

Puis la commande `gcc -S -fverbose-asm loop.c` est exécutée dans un terminal linux. Un fichier `loop.s` apparaît. L'option `-S` indique la génération du code assembleur et `-fverbose-asm` ajoute des commentaires tel que la ligne C correspondant à l'instruction. On extrait du résultat la partie qui nous intéresse :

```
;file loop.s
.L2:
# loop.c:2:       while (1);
jmp     .L2     #
```

Le compilateur gcc a bien implémenté la boucle infinie via une instruction jump indiquant un saut du PC. Dans cette exemple, la boucle étant vide, le PC saute au même endroit.

Condition sur E/S

Le lignes suivantes implémenté via une structure `if else` traduise le comportement souhaité du point de vue utilisateur. A savoir, le maintient en position enfoncé du bouton provoque l'illumination de la LED0. La lecture de son état est permis grâce à la fonction **port_pin_get_input_level** retournant un entier de valeur XX ou XX. Elle est alors comparé à `LED_0_ACTIVE` définit comme XX. Si la condition est vrai la fonction `port_pin_set_output_level` est appelé avec comme paramètre `LED_0_ACTIVE` sinon `!LED_0_ACTIVE`.

3 Programmation d'un timer

Cette étape vise à générer un signal carré de période 1ms sur une des sorties timer du microcontrôleur. Il s'agit donc de préparer l'implantation de la fonction Horloge. Cette fonction sera donc réalisée par une ressource matérielle du microcontrôleur ; un timer.

3.1 Fonctionnement d'un timer

Rappel fonctionnement d'un timer.

3.2 Configuration TC6

Configuration du TC6.

References

[1] Pompignac Hernando Monteil, Nicomette. *Du langage C au C++*. Pour l'ingénieur, 2012.

[2] ANSSI. Règles de programmation pour le développement sécurisé de logiciels en langage c, November 2021.