projet
document

Louison GOUY Yiying WEI

16 janvier 2022

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 Glossaire

Requête d'interruption : (IRQ : interrupt request) Signal matériel indiquant qu'une interruption est requise.

Polling : approche d'ordonnancement dans laquelle le logiciel répète un test sur une condition pour déterminer s'il doit exécuter une tache. [**?**]

Une **machine état ou automate fini** est une construction mathématique abstraite, susceptible d'être dans un nombre fini d'*états* , mais étant un moment donné dans un seul état à la fois. Le passage d'un état à un autre ce fait par une *transition*.

2 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 [**?**]. 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 permissif. 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* [**?**], visant à "favoriser la production de logiciels C plus sécurisés, plus sûrs, d'une plus grande robustesse et portables". Il servira de référence durant ce projet.

3 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.

3.1 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
```

La première ligne permet d'inclure la bibliothèque asf et ainsi de profiter du niveau d'abstraction mis à disposition par Microship. La suivante, `int main ()` bien connue 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 etc. 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) en "continu". La condition suivante d'éclanche le résultat souhaité : allumage ou extinction de la LED0.

3.2 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. 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. Elle sera utilisée tout au long de ce TP et fera l'objet de nombreuses références.

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.

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);}
3
```

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. Il est intéressant de faire le parallèle avec l'assembleur. Cet exemple reste toutefois approximatif puisque ce n'est pas le jeux d'instruction du CORTEXM0+ qui a été utilisé. Prenons le comme une introduction.

Condition sur E/S

Les lignes suivantes implémentées via une structure `if else` traduisent 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`.

4 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.

4.1 Fonctionnement d'un timer

Le microcontrôleur SAMD21 possède 5 timers/counters allant de TC3 à TC7. Il est possible de les paramétrer en fonction de l'utilisation qu'il en sera fait. Dans notre cas, le timer TC6 est imposé par le sujet du TP.
Chaque timer peut prendre 3 configurations possibles : 8, 16 ou 32 bits [1]. Le nombre de registres associés à chacune des configurations est différent. Nous utiliserons le mode 16 bits (65536 valeurs possibles).

Fonctionnement du TC en mode waveform
Les timers/counters (TC) du microcontrôleur SAMD21 proposent un mode de fonctionnement adapté à la production de signaux logiques : le mode *waveform*. La sélection du mode se fait via la configuration de certains registres. L'objectif est de générer un signal rectangulaire de rapport cyclique quelconque.
Il existe 4 modes de fonctionnement pour les compteurs en mode WAVEFORM présenté par la figure ci-dessous.

Name	Operation	TOP	Update	Output Waveform		OVFIF/Event	
				On Match	On Update	Up	Down
NFRQ	Normal Frequency	PER	TOP/ ZERO	Toggle	Stable	TOP	ZERO
MFRQ	Match Frequency	CC0	TOP/ ZERO	Toggle	Stable	TOP	ZERO
NPWM	Single-slope PWM	PER	TOP/ ZERO	See description above.		TOP	ZERO
MPWM	Single-slope PWM	CC0	TOP/ ZERO	Toggle	Toggle	TOP	ZERO

FIGURE 1 – Mode de fonctionnement en WAVEFORM pour les compteurs

Le mode Match Frequency Generation (MFRQ) est le plus adapté à l'application. En effet, la fréquence n'est fixée qu'avec un seul paramètre CC0. D'après la datasheet du SAMD21, le fonctionnement du mode MFRQ est le suivant.

1. Le timer 32bits fonctionne en assemblant 2 timers 16 bits en cascade

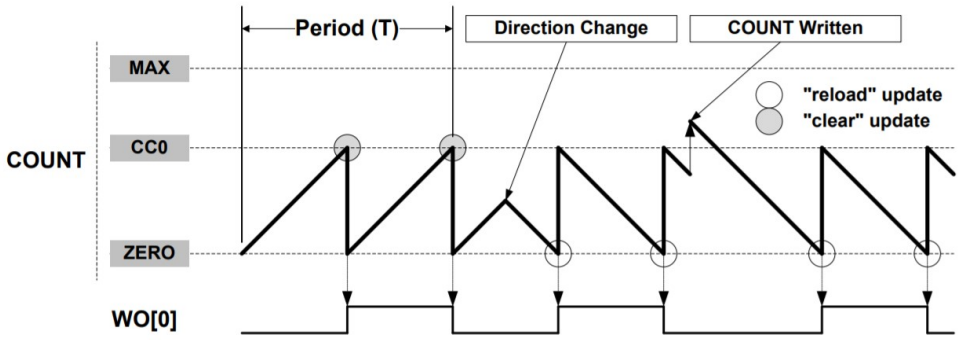

FIGURE 2 – Fonctionnement timer configuré en MFRQ

La période T du signal est contrôlée par le registre CC0. Le signal de sortie est numérique, sa valeur se trouve dans WO[0]. A chaque fois que le compteur COUNT atteint la valeur du registre CC0. Le signal de sortie WO[0] est permuté. La valeur MAX correspond à la résolution du compteur : ici 16 bits donc 65536 valeurs possibles. Il faudra être vigilent car la valeur du compteur vaut deux fois celle du signal de sortie.

Calculs pour une fréquence de 1kHz

Pour obtenir une fréquence de 1kHz il faut déterminer la valeur de CC0 comme expliqué précédemment. Pour faire cela il est primordial de bien comprendre son fonctionnement et les registres impliqués dans la configuration. La figure ci-dessous donne la fréquence de comptage.

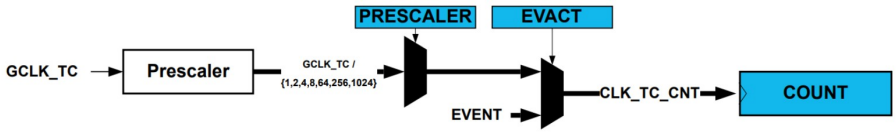

FIGURE 3 – Configuration du compteur TC

L'horloge count est fournie à partir de l'horloge GCLK_TC (Generic clock for TC). Elle est l'horloge de référence pour les TC. Elle a une fréquence de 8MHz. Cette horloge peut être divisée en y appliquant un prescaler afin d'obtenir CLK_TC_CNT. N est une pré division de l'horloge du timer. Dans notre cas, le prescaler n'est pas appliqué et prendra la valeur N = 1. L'équation ci-dessous présente la fréquence à laquelle sera effectué le comptage.

$$T_{GCLK\_TC} = N * T_{CLK\_TC\_CNT} = T_{CLK\_TC\_CNT} \tag{1}$$

Donc

$$f_{GCLK\_TC} = f_{CLK\_TC\_CNT} \tag{2}$$

La fréquence souhaitée est établie à partir de l'équation suivante :

$$f_{WO[0]} = \frac{f_{CLK\_TC\_CNT}}{2 * (CC0 + 1)} \tag{3}$$

On sait que f_{GCLK_TC} est égale à 8 MHz. Pour une fréquence $f_{WO[0]}$ de 1kHz, on obtient

$$CC0 = \frac{f_{GCLK\_TC}}{2 * f_{WO[0]}} - 1 = 3999 \tag{4}$$

La valeur chargée dans le registre CC0 sera donc 3999.

4.2 Configuration pour le TC6

Cette partie détaillé les configurations nécessaires à la génération d'un signal carré de 1kHz grâce à TC6. Les éléments suivants seront configuré : le generic clock controller, le power manager, un port d'entrée sortie et finalement TC6.

4.2.1 Configuration du Generic Clock(GCLK)

Chaque périphérique de la carte SAMD21 nécessite une horloge de fonctionnement interne. Pour notre périphérique du Timer, il s'agit de GCLK_TC6 (Generic clock for TC6).
La figure ci-dessous présente la génération des signaux de l'horloge périphérique et de l'horloge principale. Le Generic Clock Controller est composé de 9 générateurs et de multiplexeurs.

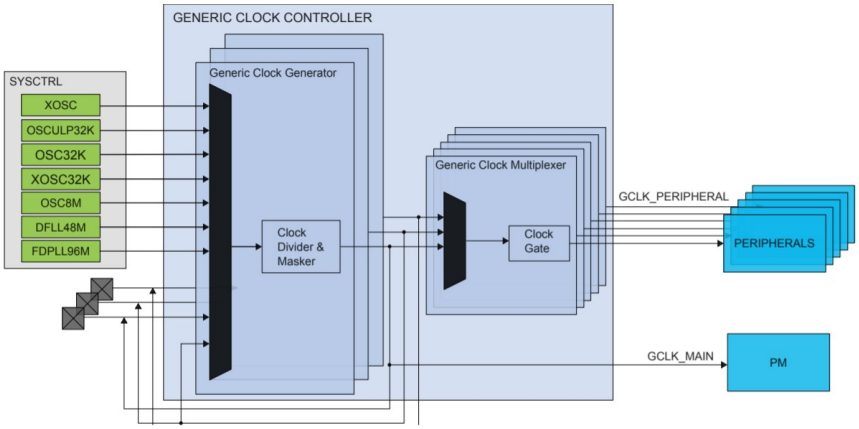

FIGURE 4 – Fonctionnement du Generic Clock Controller

On remarque que le Generic Clock Controller est divisé en deux parties. D'une part le Generic Clock Controler est configuré par le registre GENCTRL. D'autre part le Generic Clock Multiplexer est configuré par le registre CLKCTRL.

Configuration du registre GENCTRL
Le détail de Generic Clock Generator Control (GENCTRL) est donné dans la figure ci-dessous :

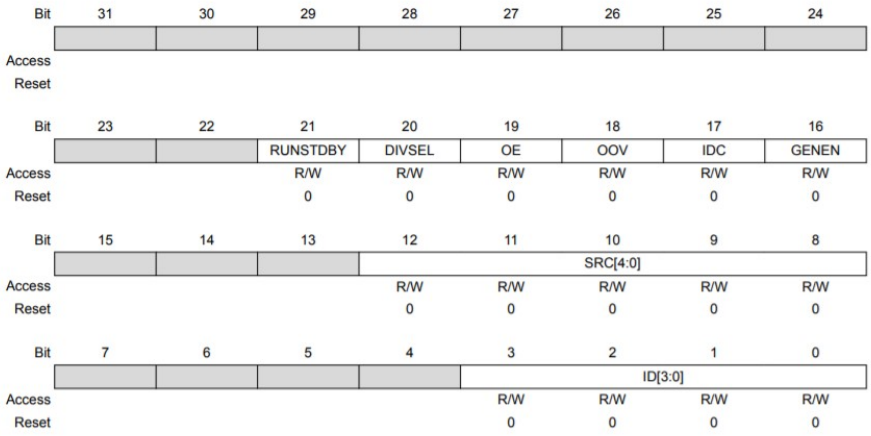

FIGURE 5 – Configuration des différents bits du GENCTRL

— **RUNSTDBY** : Fonctionnement en mode Standby ou non. Dans notre cas, nous voulons la désactiver donc il faut mettre **0<<21** dans ce champ.

— **DIVSEL** : Définit le facteur de division de l'horloge. Nous ne voulons pas la diviser donc il faut mettre la valeur **0<<20** dans ce champ.

— **OE** : Permet d'autoriser l'activation sur une sortie de GCLK. Nous ne voulons pas activer cette option donc il faut mettre la valeur **0<<19** dans ce champ.

— **OOV** : Définit la valeur de la sortie de GCLK. Lorsque l'OE est à 0 il faut mettre également 0 dans ce champ donc la valeur **0<<18**.

— **IDC** :Définit du rapport cyclique en cas de division impaire. Dans notre cas, il faut mettre la valeur **0<<17** dans ce champ.

— **GENEN** : Validation ou non du générateur d'horloge. Nous voulons l'activer donc il faut mettre la valeur **1<<16** dans ce champ.

— **SRC[4 :0]** : Choix de la source d'horloge. Nous voulons choisir la source OSC8M donc d'après la datasheet il faut mettre la valeur **6<<8** dans ce champ.

— **ID[3 :0]** : Définit le numéro du générateur que l'on configure (0 à 8). Nous choisissons la générateur 0 donc il faut mettre la valeur **0<<0**.

Configuration du registre CLKCTRL
Ce registre permet de choisir parmis les 9 générateurs décrit précédemment. Le détail de Generic Clock Control (CLKCTRL) est donné dans la figure ci-dessous :

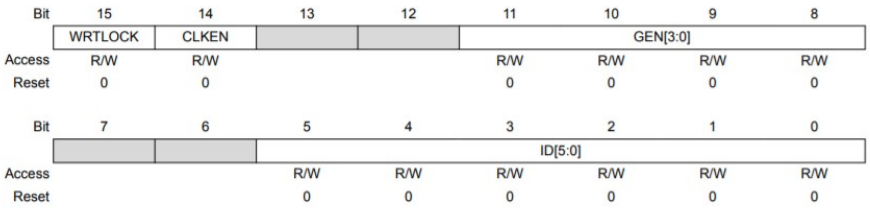

FIGURE 6 – Configuration des différents bits du CLKCTRL

— **WRTLOCK** : Permet le verrouillage de l'horloge générique pour les générateurs 1 à 9. Nous avons choisi le générateur 0 et il n'y a pas de verrouillage donc il faut mettre **0<<15** dans ce champ.

— **CLKEN** : Validation de l'horloge générique donc **1<<14** dans ce champ.

— **GEN[3 :0]** : Permet de choisir le générateur d'horloge d'entrée. Ici il s'agit GCLK_GEN[0] donc il faut mettre **0<<8** dans ce champ.

— **ID[5 :0]** : Définit le périphérique vers lequel est dirigé l'horloge générique. Dans notre cas, il faut l'envoyer vers TC6 donc d'après la datasheet du SAMD21 il faut mettre la valeur **1D<<0**.

Code de la configuration du GCLK

La configuration du GCLK traduite en langage C donne le résultat suivant :

```
1   void config_GCLK_TC6(void){
2         Gclk *ptr_GCLK = GCLK;
3
4         ptr_GCLK->CLKCTRL.reg = GCLK_CLKCTRL_NOWRTLOCK |
5                                 GCLK_CLKCTRL_CLKEN |
6                                 GCLK_CLKCTRL_GEN_GCLK0 |
7                                 GCLK_CLKCTRL_ID_TC6_TC7 ;
8
9         ptr_GCLK->GENCTRL.reg = GCLK_GENCTRL_NORUNSTDBY |
10                                GCLK_GENCTRL_NODIVSEL |
11                                GCLK_GENCTRL_NOOE |
12                                GCLK_GENCTRL_NOOOV |
13                                GCLK_GENCTRL_NOIDC |
14                                GCLK_GENCTRL_GENEN |
15                                GCLK_GENCTRL_SRC_OSC8M |
16                                GCLK_GENCTRL_ID(0);
17  }
18
```

4.2.2 Configuration du Power Manager(PM)

Une horloge "bus" pour le timer TC6 est délivrée par le Power Manager (PM). Cette horloge permet du dialogue entre le Microprocesseur et le périphérique.
Le Power Manager montré en figure ci-dessous gère une clock pour le CPU, une clock pour le bus AHB à destination de la mémoire et trois clocks pour le bus APB à destination des périphériques.

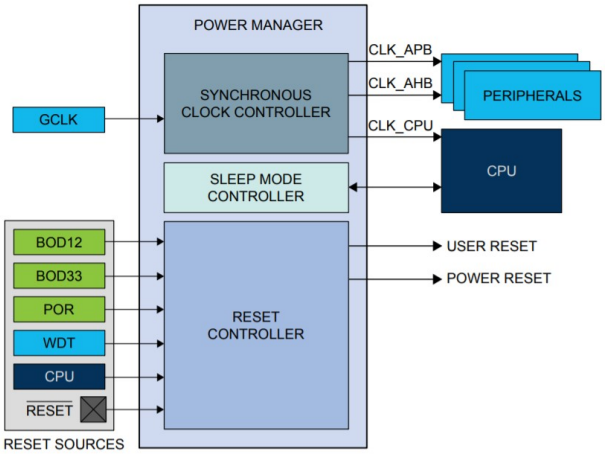

FIGURE 7 – Structure du Power Manager

On programme le "Synchronous Clock Controller" en entré CLK, en sortie 3 horloges par les 3 bus interes du Microprocesseur : APBA, APBB et APBC.
D'après la cartographie des produits du datasheet, le Timer TC6 est place sur le bus APBC. Donc il faut configurer les registres :

— APBCSEL
— APBCMASK
— CPUSEL

Configuration du registre APBCSEL
Le registre APBA Clock Select (APBCSEL) ne contient qu'un seul champ :

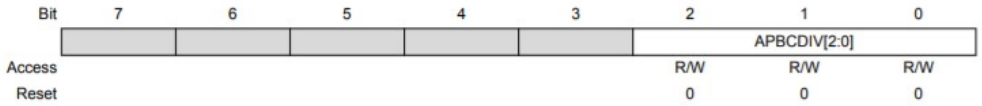

FIGURE 8 – Configuration des différents bits du APBCSEL

— **APBCDIV[2 :0]** : Définit le facteur de division de l'horloge d'entrée GCLKMAIN. On choisit la division par 1 donc il faut mettre **0<<0**.

Configuration du registre APBCMASK
Le registre APBC Mask (APBCMASK) contient les validations d'horloge bus pour tous les périphériques connectes sur le bus APBC.

Bit	31	30	29	28	27	26	25	24
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
				I2S	PTC	DAC	AC	ADC
Access	R	R	R	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	1

Bit	15	14	13	12	11	10	9	8
	TC7	TC6	TC5	TC4	TC3	TCC2	TCC1	TCC0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	SERCOM5	SERCOM4	SERCOM3	SERCOM2	SERCOM1	SERCOM0	EVSYS	PAC2
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

FIGURE 9 – Configuration des différents bits du APBCMASK

Les bits 20 :0 permettent de stopper les différentes horloges bus de APBC s'ils sont mis à zéro ou bien de les activer s'ils sont mis à un.

— **TC6** : Ici on veut activer l'horloge bus pour TC6 donc il faut mettre **1<<14**.

Configuration du registre CPUCSEL

Le registre CPU Clock Select (CPUSEL) ne contient qu'un seul champ :

Bit	7	6	5	4	3	2	1	0
						CPUDIV[2:0]		
Access						R/W	R/W	R/W
Reset						0	0	0

FIGURE 10 – Configuration des différents bits du CPUSEL

— **CPUDIV[2 :0]** : Permet de définir de facteur de division de l'horloge du CPU par rapport à GCLKMAIN. On choisit la division par 1 donc il faut mettre **0<<0**.

Code de la configuration du PM

La configuration du PM traduite en langage C donne le résultat suivant :

```
1 void config_PM(void){
2       Pm *ptr_PM = PM;
3
4       ptr_PM -> CPUSEL.reg = PM_CPUSEL_CPUDIV_DIV1;
5       ptr_PM -> APBCSEL.reg = PM_APBCSEL_APBCDIV_DIV1;
6       ptr_PM -> APBCMASK.reg |= PM_APBCMASK_TC6;
7 }
8
```

4.2.3 Configuration des ports d'E/S parallèles

Le nombre de pattes du microprocesseur est limité : le SAMD21 possède 64 pattes, un nombre bien inférieur à la somme des entrées/sorties de tous les périphériques.
Il faut donc multiplexer les entrées ou les sorties des périphériques via les ports d'entrée/sortie A et B.
Dans un premier il nous faut donc trouver sur quelles pins, et dans quel multiplexage il est possible d'observer le signal WO[0] du TC6. D'après la table de multiplexage présente dans la datasheet du SAMD21, ce signal est disponible sur 2 pins : PB02 et PB16, en multiplexage **E**. Cependant la pin PB16 est utilisé aussi pour d'autre fonction, il est donc préférable de prendre la PB02.
Pour accéder a la sortie WO[0] de TC6, il faudra configurer :

— Registre **DIRSET** pour choisir la broche PB02 en sortie
— Registre **PMUX** pour choisir le multiplexage de type E
— Registre **PINCFG** pour valider le multiplexage choisi

Configuration du registre DIRSET
Le registre Data Direction Set (DIRSET) permet à l'utilisateur de définir une ou plusieurs broches d'E/S en sortie.
Ce registre sert à mettre à 1 en bits correspondants dans le registre DIR. Le registre DIR contient le configuration de chacune des pattes du port A ou B.

Bit	31	30	29	28	27	26	25	24
	DIRSET[31:24]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	23	22	21	20	19	18	17	16
	DIRSET[23:16]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8
	DIRSET[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	DIRSET[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

FIGURE 11 – Configuration des différents bits du CPUSEL

— **DIRSET[31 :0]** : Le bit à 0 n'a aucun effet. Le bit à 1 configure la broche d'E/S comme une sortie. Pour le port B, il faut configurer la patte 2 en sortie donc il faut mettre **1<<2**.

Configuration du registre PMUX
Il y a jusqu'à 16 registres de multiplexage périphérique dans chaque groupe, un pour chaque ensemble de deux lignes d'E/S. Le n désigne le numéro de l'ensemble des lignes d'E/S.

Offset	Name	Bit Pos.								
0x2C ... 0x2F	Reserved									
0x30	PMUX0	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x31	PMUX1	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x32	PMUX2	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x33	PMUX3	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x34	PMUX4	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x35	PMUX5	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x36	PMUX6	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x37	PMUX7	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x38	PMUX8	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x39	PMUX9	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x3A	PMUX10	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x3B	PMUX11	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x3C	PMUX12	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x3D	PMUX13	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x3E	PMUX14	7:0	PMUXO[3:0]				PMUXE[3:0]			
0x3F	PMUX15	7:0	PMUXO[3:0]				PMUXE[3:0]			

FIGURE 12 – Registre du PMUXn

Le registre Peripheral Multiplexing n (PMUXn) est composé de 16 octets, chacun composé de 2 parties : 4 bits pour les pins impaires, 4 bits pour les pins paires. Ainsi PMUX0 contient le multiplexage des pins 0 (paire) et 1 (impaire). La pin 2 est alors la partie paire de PMUX1.

Bit	7	6	5	4	3	2	1	0
	PMUXO[3:0]				PMUXE[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

FIGURE 13 – Configuration des différents bits du PMUXn

— **PMUXE[3 :0]** : 0×0 étant le multiplexage A, et 0×8 le I. On spécifie alors le multiplexage à mettre en place : E, donc il faut mettre la valeur **0×4<<0**.

Configuration du registre PINCFG

Enfin, il faut activer la pin PB02 en passant par les registres Pin Configuration (PINCFG). Ces registres sont définis comme suit :

Bit	7	6	5	4	3	2	1	0
		DRVSTR				PULLEN	INEN	PMUXEN
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

FIGURE 14 – Configuration des différents bits du PINCFGn

— **DRVSTR** : Contrôle la force du driver de sortie. 1 pour fort, 0 pour normal. Ne nous concerne pas ici donc il faut mettre **0<<6** dans ce champ.

— **PULLEN** : Active ou non un la résistance pull-up ou pull-down interne d'une broche d'E/S configurée en entrée. Ne nous concerne pas ici, donc **0<<2**.

— **INEN** : : Validation de la patte comme une entrée. On considère ici une sortie donc il faut mettre **0<<1** dans ce champ.

— **PMUXEN** : Validation ou non le multiplexage E mis en place par le registre PMUX correpondant donc il faut mettre **1<<0**.

Code de la configuration du PORT
La configuration du PORT traduite en langage C donne le résultat suivant :

```
void config_PORT(void){
      Port *ptr_port = PORT ;

      ptr_port -> Group[1].DIRSET.reg = PORT_PB02;
      ptr_port -> Group[1].PMUX[2/2].reg = PORT_PMUX_PMUXE_E;
      ptr_port -> Group[1].PINCFG[2].reg = PORT_PINCFG_DRVSTR_NO |
                                           PORT_PINCFG_PULLEN_NO |
                                           PORT_PINCFG_INEN_NO |
                                           PORT_PINCFG_PMUXEN;
}

```

4.2.4 Configuration du périphérique Timer Counter

Nous utiliserons pour cette partie le Timer Counter 6 (TC6) configuré dans le mode 16 bits. Pour générer un signal de fréquence 1kHz, il faut programmer le registre :
— CTRLA
— CTRLBLR
— CTRLC
— CC0

Configuration du registre CTRLA
Le détail de Control A (CTRLA) est donné dans la figure ci-dessous :

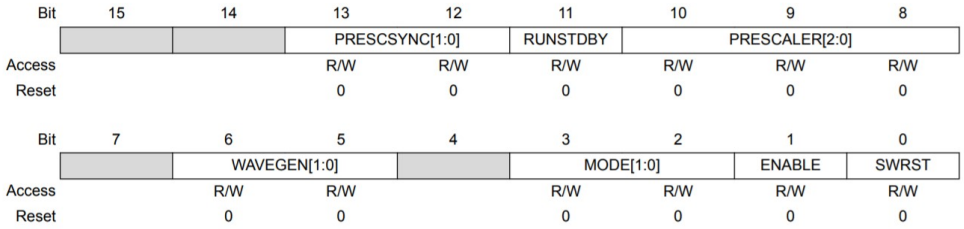

FIGURE 15 – Configuration des différents bits du CTRLA

— **PRESCSYNC[1 :0]** : Le registre de comptage est piloté par une horloge issue de GCLK_TC via un prescaler. Il faut donc mettre la valeur **1<<12**.

— **RUNSTDBY** : Définit le fonctionnement du TC en mode STANDBY. Il n'y a pas d'intérêt à garder le TC actif en veille donc **0<<11**.

— **PRESCALER[2 :0]** : Applique un facteur de division à l'horloge d'entrée entre 1 et 1024. Il ne sera pas nécessaire de diviser au préalable l'horloge donc on garde un prescaler de 1 alors **0<<8**.

— **WAVEGEN[1 :0]** : Sélectionne le mode de fonctionnement du compteur. Le mode MFRQ est utilisé, il faut donc **1<<5**.

— **MODE[1 :0]** : Sélectionne le mode de comptage pour le compteur, c'est-àdire le nombre de bits de comptage : 8, 16 ou 32 bits. Il nous faut ici compter 4000 valeurs (de 0 à 3999) d'après le modèle présenté en début de section. C'est donc le mode 16 bits qui est choisi, alors **0<<2**.

— **ENABLE** : Valide le timer ou non. Ici nous voulons le valider donc il faut mettre la valeur **1<<1**.

— **SWRST** : Nous ne voulons pas de RESET du Timer donc il faut mettre la valeur **0<<0**.

Configuration du registre CTRLBCLR
Le détail de Control B Clear (CTRLBCLR) est donné dans la figure ci-dessous :

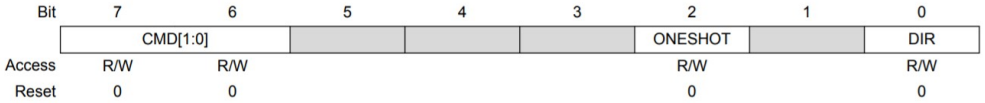

FIGURE 16 – Configuration des différents bits du CTRLBCLR

— **CMD[1 :0]** : Sélectionne la commande lors du prochain cycle de GCLK du TC : NONE, RETRIGGER ou STOP. Aucun comportement particulier n'est souhaité, on place alors NONE **0<<6**.

— **ONESHOT** : Le One-Shot stop le compteur lors d'un débordement du compteur (inférieur à 0 ou supérieur à la valeur maximale). Placé ce bit à 1 stop cette fonctionnalité. On veut le désactiver donc il faut mettre la valeur **1<<2**.

— **DIR** : Paramètre le sens de comptage : incrémentation(1) ou décrémentation(0). On choisit le mode incrémention donc **1<<0**.

Configuration du registre CTRLC
Le détail de Control C (CTRLC) est donné dans la figure ci-dessous :

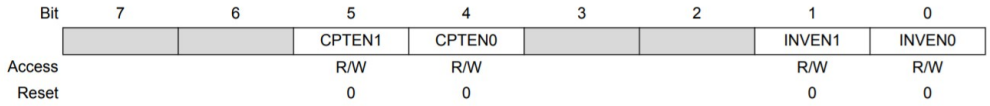

FIGURE 17 – Configuration des différents bits du CTRLC

— **CPTENx** : Autorise la capture sur les channel 1 et 0. On veut désactiver le mode capture 1 et capture 0 donc il faut mettre la valeur **0<<5** et **0<<4**.

— **INVENx** : Inverse les sorties WO[1] et WO[0] lorsque ces bits sont à 1. On ne veut pas inverser la sortie WO[1] et WO[0] donc il faut mettre la valeur **0<<1** et **0<<0**.

Configuration du registre CC0
Ce registre permet de définir la demi période CC0 du signal W0[0].
D'après les calculs présenté précédemment, pour générer un signal de fréquence 1kHz, la valeur chargée dans le registre CC0 sera 3999.

Code de la configuration du TC6
La configuration du TC6 traduite en langage C donne le résultat suivant :

```
1  void config_TC6(void){
2        Tc *ptr_TC = TC6 ;
3
4        ptr_TC -> COUNT16.CTRLA.reg = TC_CTRLA_PRESCSYNC_PRESC |
5                                      TC_CTRLA_NORUNSTDBY |
6                                      TC_CTRLA_PRESCALER_DIV1 |
7                                      TC_CTRLA_WAVEGEN_MFRQ |
8                                      TC_CTRLA_MODE_COUNT16 |
9                                      TC_CTRLA_ENABLE |
10                                     TC_CTRLA_NOSWRST;
11       ptr_TC -> COUNT16.CTRLC.reg = TC_CTRLC_NOCPTEN0 |
12                                     TC_CTRLC_NOCPTEN1 |
13                                     TC_CTRLC_NOINVEN0 |
14                                     TC_CTRLC_NOINVEN1;
15       ptr_TC -> COUNT16.CTRLBCLR.reg = TC_CTRLBCLR_CMD_NONE |
16                                        TC_CTRLBCLR_ONESHOT |
17                                        TC_CTRLBCLR_DIR;
18       ptr_TC -> COUNT16.CC[0].reg = 3999;
19       ptr_TC -> COUNT16.CTRLA.reg |= TC_CTRLA_ENABLE;
20 }
21
```

4.3 Test et validation

Maintenant que le programmes est terminé, il est implanté sur la carte.
A l'aide d'un oscilloscope, on visualise le signal de sortie sur la pin PB02 paramétrée précédemment.
Le signal en sortie du TC6 présenté en figure ci-dessous :

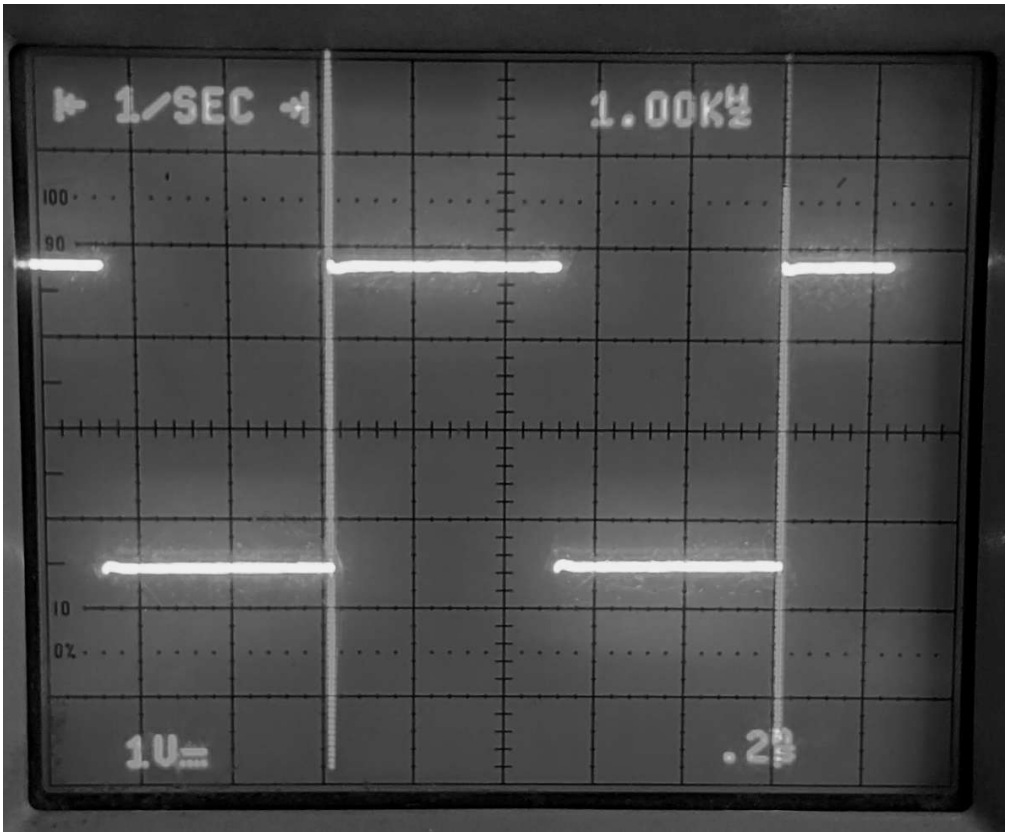

Figure 18 – Signal de sortie du TC6

On obtient bien un signal carré de fréquence 1kHz comme signal en sortie du TC6.

5 Fonction sous interruption

Les interruptions sont des outils essentiels pour concevoir des systèmes réactifs (en : responsive) dès lors qu'ils doivent exécuter des opérations logicielles et matérielles simultanément. Avant de présenter le fonctionnement et la configuration du système d'interruption du Cortex-M il est préférable de donner un aperçu du problème.

5.1 Présentation du problème

L'environnement avec lequel interagi le microcontrôleur est dit asynchrone. Il ne peut à priori par connaître l'instant d'apparition d'un événement. Il est alors obligé de le scruter fréquemment pour être averti rapidement d'un changement. C'est précisément ce que fait l'étape 1. La lecture du bouton est placée dans une boucle infinie, si l'utilisateur appui sur ce denier l'état de la led est modifié. Bien que ce programme soit inefficace, il fonctionne bien et offre une faible latence. Ce mode de fonctionnement est appelé "polling". Le problème survient dès lors qu'une tâche logicielle est ajoutée au programme. Le processeur n'est plus seulement occupé à contrôler l'état du bouton mais également à exécuter une autre partie du programme. La réactivité n'est alors plus garantie. Dans certains cas, si la tâche logicielle est importante, le changement d'état du bouton pourra être invisible pour le programme.

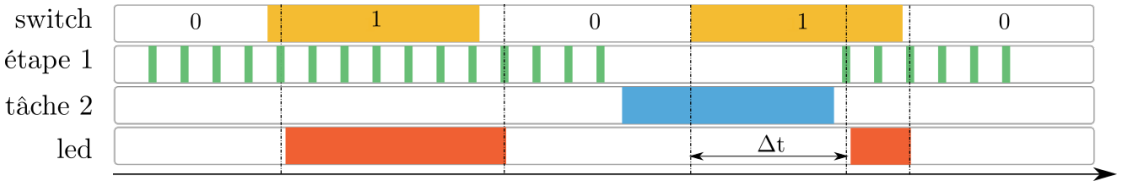

FIGURE 19 – Ordonnancement tâches avec polling

La figure ci-dessus représente l'exécution [2] d'un programme comme celui de l'étape 1 auquel une seconde tâche a été ajoutée (tâche 2). Deux cas sont illustrés dans cet exemple. Le premier, à gauche, correspond au bon déroulement, le processeur lit de manière successive l'état du switch (étape 1), lorsque une lecture vaut '1' l'état de la led est mis à '1' et même chose pour '0'. Une petite (latence) est présente mais elle sera généralement négligée. Dans le second cas, à droite, la tâche 2 est en cours d'exécution est empêche la lecture du bouton. Une fois terminée la mise à jours de la led peut s'opérer comme précédement. L'exécution de la tâche 2 a générer un retard qui n'est cette fois-ci plus négligeable du point de vue de l'utilisateur. Si la tâche 2 avait durée un instant de plus le système n'aurait pas réagi à la stimulation. Un tel cas ne doit, en aucun cas, se produire pour un système dont la sécurité est critique.
En pratique ce problème survient fréquemment. Les interactions utilisateurs, tel que l'appuis sur un bouton, sont relativement lentes comparés à la vitesse de fonctionnement des processeurs. Un "polling" pourra suffire dans certains cas. Mais pour ce qui est des événements matériels, comme un timer ou un protocole série, la situation est plus critique. Il n'y a alors pas d'autres solutions que d'utiliser une interruption. De manière générale les logiciels utilisant des interruptions fonctionnent plus efficacement que ceux basés sur le "pooling". Aucun temps n'est perdu à contrôler l'apparition d'un événement. L'approche par déclenchement sur évènements (en : even-trigger) offre également une meilleur réactivité. [**?**]

2. Aucune échelle précise n'est choisi le but étant une simple présentation fonctionnelle.

5.2 Fonctionnement d'une interruption

Une interruption est définie comme une suspension temporaire de l'exécution d'un programme informatique par le microprocesseur afin d'exécuter un programme prioritaire (routine d'interruption). La dite suspension est déclenchée par une source externe.
La gestion d'une interruption s'effectue comme tel : pause du programme, sauvegarde du contexte, identification de la routine d'interruption à exécuter (en fonction de ma source), exécution de la routine et rétablissent du contexte. Ces actions sont gérées par le NVIC, Nest Vectored Interrupt Controler faisant l'interface entre les périphériques et le CPU.

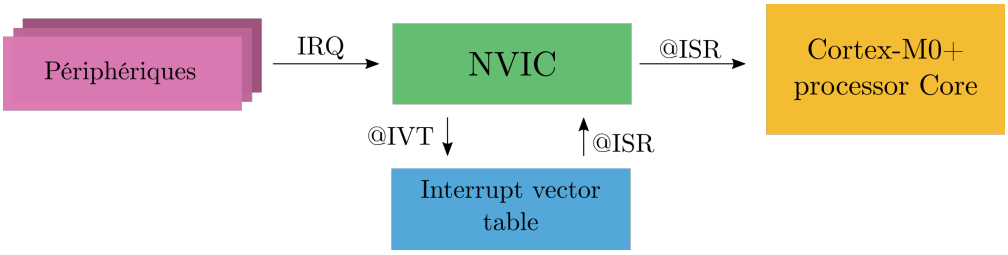

FIGURE 20 – Schéma bloc NVIC

Lorsqu'une interruption x se produit, la demande d'interruption (IRQ) est envoyée au NVIC. Si le NVIC accepte la demande, l'étape suivante du NVIC consiste à trouver l'adresse de départ de la routine d'interruption (@ISR). Cette adresse est stockée dans la table de vecteurs d'interruption (IVT). Le NVIC utilise le numéro d'interruption x pour calculer l'adresse de l'exception dans la table des vecteurs d'interruption. Il utilise ensuite le contenu de cette adresse mémoire pour exécuter le gestionnaire d'exception. Le compteur de programme est alors chargé avec l'adresse de l'ISR et le CPU commence à exécuter la routine d'exception. Un rappel du vocabulaire utilisé dans la figure 20.

@	adresse mémoire
IRQ	Interrupt Service Routine
NVIC	Nest Vectored Interrupt Controler
IVT	Interrupt Vector table
ISR	Interrupt Service Rotine

TABLE 1 – Terminologie mécanisme d'interruption

La table des vecteurs d'interruptions, est une structure de données et, comme son nom l'indique, un tableau contenant des vecteurs. En programmation embarquée, un vecteur désigne une adresse mémoire. Par conséquent, une table de vecteurs est une table contenant des adresses mémoire. **La table des vecteurs d'interruption contient les adresses (pointeurs de fonction) des routines d'interruption et des fonctions de gestion des exceptions.** Les exceptions désignent les évènements en mesure d'interrompre le déroulement normal du programme pour effectuer une courte tache. Les interruptions sont donc partie des exceptions. Elles ne sont pas les seules le processeur lui même peut déclencher le mécanisme d'exception lorsque on lui demande de faire quelque chose qu'il ne parvient pas à faire. Par exemple la lecture d'un mot de 32 bits à une adresse impaire provoquera une exception à l'exécution.

5.3 Configuration interruption

Pour qu'une interruption se produise il faut que le périphérique soit configuré de sorte à réagir à l'occurrence d'un événements spécifique. Dans le contexte du projet nous devons calculer le gabarit toutes les 1 ms. Cette partie présente la configuration à effectuer pour que le timer TC6 déclenche une routine d'interruption contenant la fonction calcul gabarit. Elle détaillera également l'autorisation de l'interruption par le NVIC.

5.3.1 Timer TC6

L'activation de l'interruption sur le périphérique TC6 est documentée en détaille dans la datasheet. [?] Chaque source d'interruption est associée à un drapeau d'interruption. Le drapeau d'interruption (en : interrupt flag), dans le registre (INTFLAG), est activé lorsque la condition d'interruption en vrai. Chaque interruption peut être activée individuellement en écrivant un '1' au bit correspondant dans le registre d'activation Interrupt Enable Set (INTENSET) et désactivée en écrivant un '1' au bit correspondant dans le registre de désactivation Interrupt Enable Clear (INTENCLR). Le registre INTENSET se présente comme tel :

Bit	7	6	5	4	3	2	1	0
			MC1	MC0	SYNCRDY		ERR	OVF
Access			R/W	R/W	R/W		R/W	R/W
Reset			0	0	0		0	0

FIGURE 21 – Registre INTENSET

Ayant programmé le timer en mode MFRQ dans la partie précédente, seul le bit 4 nous intéresse. Il est chargé à '1' pour activer l'interruption sur correspondance de COUNT et MC0. L'implémentation en C est intuitive :

```
1  void configure_TC6_IT(void){
2    Tc *ptr_TC = TC6;
3    ptr_TC->COUNT16.INTENSET.reg = TC_INTFLAG_MC0;
4  }
5
```

Comme l'indique la datasheet après la configuration du périphérique il faut activer l'interruptions correspondante dans NVIC.

> "The interrupt request line is connected to the Interrupt Controller. In order to use interrupt requests of this peripheral, the Interrupt Controller (NVIC) must be configured first." [?](page 557)

5.3.2 NVIC

Le NVIC est un élément interne au processeur. La documentation de Microchip donne pour chaque périphérique le numéro d'interruption associée mais renvoit à la documentation de Arm pour la configuration du NVIC. Le numéro du TC6 est 21. Comme nous avons pu l'évoquer plus haut le NVIC peut accepter ou non les interruptions provenant des périphériques. Il consulte pour cela un registre nommé NVIC_ISER contenant un seul champ de 32 bits SETENA littéralement set enable. L'extrait de la documentation [?] Arm ci-dessous nous donne la valeur des bits à charger.

Bits	Name	Function		
[31:0]	SETENA	Enables, or reads the enabled state of one or more interrupts. Each bit corresponds to the same numbered interrupt:		
		On reads	**0**	the associated interrupt is disabled.
			1	the associated interrupt is enabled.
		On writes	**0**	no effect.
			1	enable the associated interrupt.

FIGURE 22 – Registre NVIC_ISER ARMv6-M

Ce registre permet de lire l'activation ou d'activer l'interruption sur chaque ligne. A noter : il faut utiliser un autre regiqtre pour la désactivation, écrire un '0' n'aura aucun effet. **Nous devons dans notre cas placer un '1' sur le bit 21.** Cette configuration est grandement simplifiée par l'ASF. Mais il est essentiel de comprendre et de retenir qu'une activation supplémentaire au périphérique est nécessaire. Le code ajouter à la fonction main est le suivant :

```
1     system_interrupt_enable(SYSTEM_INTERRUPT_MODULE_TC6);
```

L'appel de cette fonction est simple et permet l'abstraction du matériel. Le paramètre passé est le numéro d'interruption `TC6_IRQn` comme le montre la définition dans `system_interrupt_features.h`

```
1   SYSTEM_INTERRUPT_MODULE_TC6 = TC6_IRQn
```

`TC6_IRQn` est défini plus haut et dépend de la cible utilisée dans notre cas le SAMD21J18A.

```
1     TC6_IRQn                  = 21
```

Sa valeur est bien 21. Nous venons d'observer la cohérence entre la datasheet et l'ASF. Pour finaliser le raisonnement regardeons l'implémentation de la fonction `system_interrupt_enable()`.

```
1   static inline void system_interrupt_enable(
2   const enum system_interrupt_vector vector)
3   {
4     NVIC->ISER[0] = (uint32_t)(1 << ((uint32_t)vector & 0x0000001f));
5   }
```

Cette fonction est particulièrement intéressante d'un point de vue pédagogique par l'utilisation de static, inline et du masquage sur le vecteur. D'un point de vue fonctionnel, elle place un '1' décalé du numéro d'interruption sur le registre ISER de NVIC. C'est précisément ce qui doit être fait pour activer les interruptions comme le montre le figure 22.

5.3.3 Routine

La routine d'interruption est la fonction appelée automatiquement par le NVIC lors d'un événement. L'adresse stocké à l'indice `TC6_IRQn` de la table des vecteurs d'interruption coreposndant à cette fonction est chargée dans PC. Elle est exécutée puis le PC retourne à sa valeur d'origine. La définition de `TC6_Handler` est réalisée par l'Asf, il n'est pas nécessaire de déclarer son prototype.
Dans la suite du projet la fonction calcul gabarit est exécutée par la routine d'interruption. N'étant pas encore développée, le changement d'état de PB6, jouera le rôle de témoin pour valider le bon déclenchement toutes les 1ms.

```
1   void procedure_traitement_IT(void){
2     Port *ptr_Port = PORT;
3     ptr_Port->Group[1].OUTTGL.reg = PORT_PB06;
4   }
```

```
5
6   void TC6_Handler(void){
7     Tc *ptr_TC = TC6;
8
9     procedure_traitement_IT();
10
11    ptr_TC->COUNT16.INTFLAG.reg = TC_INTFLAG_MC0;
12  }
13
```

Cette fonction d'interruption témoin commence par la déclaration du pointeur TC6. Mais si cette variable ne sert qu'à la dernière ligne il est impératif de l'initialisation au début de la fonction comme l'indique la recommandation 53 de guide de référence de l'ANSSI [**?**]. Ensuite, la procédure de traitement est appelée. Elle consiste à inverser (en :toggle) l'état du pin PB6. Pour finir, le drapeau d'interruption est remis à zéro grâce au registre Flag Status and Clear (INTFLAG) dont la structure est identique à celle de la figure 21. Il peut être intéressant de noter que la fonction `procedure_traitement_IT();` n'étant appelée qu'une seule fois dans le programme sera insérée directement dans le code en assembler. C'est l'équivalent d'un `inline` réalisé automatiquement par le compilateur.
ajouter code assembler pour le prouver et capture d'ecran frequence IT

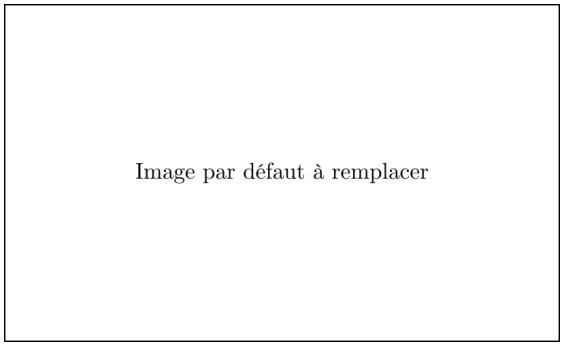

FIGURE 23 – Oscillogramme vérification procédure interruption

La fréquence du signal observé est de XXHz.

Cette partie a permis de soulever un problème lié à une programmation conjointe logiciel matériel et de proposer un solution, les interruptions. Le fonctionnement globale du système d'interruption des microcontrôleurs basés sur un Cortex-M a pu être détaillé. La configuration à travers un exemple concret utilisant le TC6 a finalement illustré ces explications.

6 Implémentation fonction calcul gabarit

Cette partie présente la conception de la fonction calcul gabarit conformément au cahier des charges donné dans le sujet. Elle a pour fonction de déterminer la vitesse de rotation du moteur, comprise entre 0 et 5000 tr/min, en fonction du temps. Le gabarit se décompose en trois phases :

— accélération d'une durée de 128 ms
— vitesse constante d'une durée de 512 ms
— décélération d'une durée de 128 ms

La valeur du palier correspondant à la vitesse constante est un paramètre variable fournis par un bloc externe avec une précision de 10 tr/min. Pour simplifier le problème elle ne pourra être modifiée au cours d'un cycle. A ce niveau dans le projet les blocs externes ne sont pas encore développés. Pour facilité l'observation lors de tests le gabarit sera généré en boucle. Un état Repos doit être ajouté pour les délimiter, sa durée sera de 64 ms.

6.1 Conception fonctionnelle

L'étape de conception fonctionnelle a pour objectif de définir une solution en ne considérant que l'aspect fonctionnel et sans tenir compte des contraintes technologiques. Il est donc question de trouver une solution en mesure de générer **le gabarit constitué des quatre étapes : repos, accélération, constante et décélération.** Ainsi la description sous forme d'un automate à état finis s'impose par la séquentialité du problème. La traduction donne le résultat suivant :

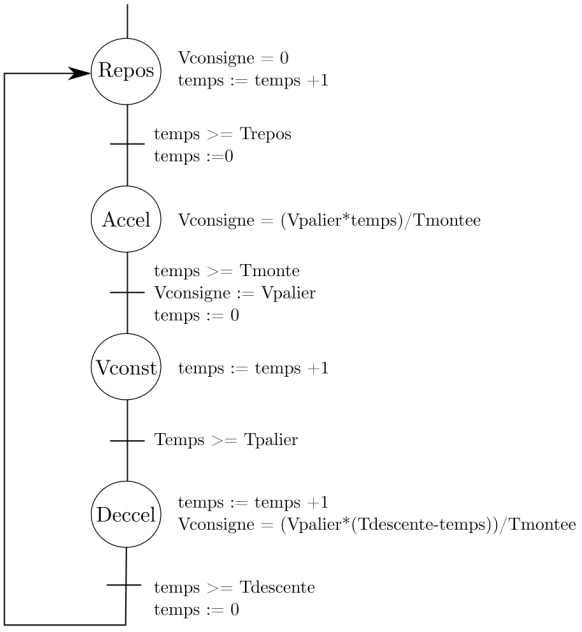

FIGURE 24 – Description fonctionnelle calcul gabarit (FSM)

Cet automate est constitué des quatre états repos, accélération, constante et décélération et de leur transition respective. La variable `Vpalier` correspond au paramètre d'entrée de la fonction. Le résultat de l'exécution, placé dans `Vconsigne`, devra être communiqué au reste du système.

6.2 Implémentation logicielle

La conception fonctionnelle précédente a permis de clarifié le problème. La première étape de la traduction en C est la déclaration des constantes. Elle se fait par l'instruction préprocesseur define ayant pour fonction de remplacer les instances suivantes de l'identifieur par la suite de lexèmes [3]. [**?**]

```
#define identifieur suite-de-lexemes
```

Ce procédé est utilisé pour définir les constantes utilisées ci-dessous. Le code C est le suivant :

```
#define TMontee 128
#define TPalier 512
#define TDescente 128
#define TRepos 64
```

Les valeurs numériques proviennent directement du cahier des charges.

Avant de traduire l'automate figure 24 il est important de définir précisément le type des variables utilisées.

nom	min	max	type générique	type C
temps	0	512	entier	uint16_t
Vconsigne	0	5000	entier	uint16_t
Vpalier	0	5000	entier	uint16_t

TABLE 2 – Types des variables fonction calcul gabarit

Le tableau ci dessus rassemble les informations permettant de choisir efficacement les types des variables. Elles sont toutes les trois entières, leur valeur minimum est supérieur à zéro et leur maximum compris entre 255 et 65535. Le type C correspondant est donc nécessairement uint16_t. Il existe une autre variable implicite à l'implémentation d'un automate. C'est l'état courant `state` de type énumération. Il peut prendre pour valeur chacun des états repos, accélération, constante et décélération. Sa définition en langage C est la suivante :

```
enum state_t{Repos, Accel, Vconst, Decel};
```

Le type énumération correspond à un entier dont la taille peut contenir le nombre d'élément. Par défaut le premier élément vaut 0, et les suivants sont supérieures de 1 par rapport à la valeur de la constante précédente. Il est possible d'affecter une valeur manuellement. Le guide des bonnes de pratiques de l'ANSSI [**?**] indique par la règle 68 qu'il ne faut pas mélanger des constantes explicites et implicites dans une énumération pour éviter tout comportement indéterminé. Toute tentative d'affectation à une valeur non définie dans l'énumération provoquera une erreur à la compilation. Dans notre cas nous laisserons la définition à la charge du compilateur. Tel qu'est construit la machine état il est nécessaire de sauvegarder les valeurs des variables d'une exécution à l'autre. Il faudra donc ajouter le qualificateur `static`.

Une fois les variables et constantes déclarées le corps de la fonction peut être implémenté. Il convient de convertir la machine état figure 24 en langage C. Cela se fait simplement autour d'un "`switch case`". Il est nécessaire de déclarer la variable `state` de type `state_t`. Un cas par défaut a été ajouté à la description fonctionnelle pour couvrir les cas d'erreur. Lors d'une exécution normale la variable `state` ne peut prendre d'autres états que ceux déclarés dans l'énumération. Si toutefois, pour une raison inconnue, elle venait à prendre une autres valeurs, le programme se trouverait dans un état indéterminé et y resterait. Avec le cas défaut le programme peut se trouver dans un état non définit, mais il sera alors remis dans son état d'origine et l'exécution pourra continuée. L'implémentation en C est la suivante :

3. Unité minimale de signification appartenant au lexique (Larousse)

```c
/* file : calcul_gabarit.c */

uint16_t calcul_gabarit (uint16_t VPalier){

  static uint16_t Temps = 0;
  static enum state_t state = Repos;
  static uint16_t Vconsigne = 0;

  switch (state)
  {
    case Repos :
      Vconsigne = 0;
      Temps ++;
      if(Temps >= 64){
      state = Accel;
      Temps = 0;
      }
      break;

    case Accel :
      if(Temps>=TMontee){
      state = Vconst;
      Temps = 0;
      Vconsigne = VPalier;
      }
      else{
      Temps ++;
      Vconsigne = (VPalier * Temps) / TMontee;
      }
      break;

    case Vconst :
      if (Temps>=TPalier){
      state = Decel;
      Temps = 0;
      }
      else{
      Temps ++;
      }
      break;

    case Decel:
      if(Temps >= TDescente){
      state = Repos;
      Temps = 0;
      Vconsigne = 0;
      }
      else{
      Temps ++;
      Vconsigne = (VPalier * (TDescente - Temps)) / TDescente;
      }
      break;

    default :
      state = Repos;
      Temps = 0;
      Vconsigne =0;
      break;
    }
    return Vconsigne;
  }
}
```

6.3 Test calcul gabarit et configuration DAC

La fonction calcul gabarit a été implémenté et doit donc être testée. Le résultat retourné est `Vconsigne`, dont la valeur analogique est comprise entre 0 et 5000. Pour visualiser son évolution dans le temps on peut s'appuyer sur un périphérique présent dans le SAMD21, le convertisseur numérique vers analogique (DAC : Digital Analog Converter). Ainsi il sera possible de visualiser le gabarit générer sous sa forme temporel tel qu'il a été décrit initialement dans le cahier des charges. Le DAC converti une valeur numérique en tension. Celui présent dans le microcontrôleur utilisé dispose d'une voix 10 bits sur 3,3V, soit une précision de 3,32mV. Sa représentation structurelle sous forme d'un diagramme bloc est la suivante :

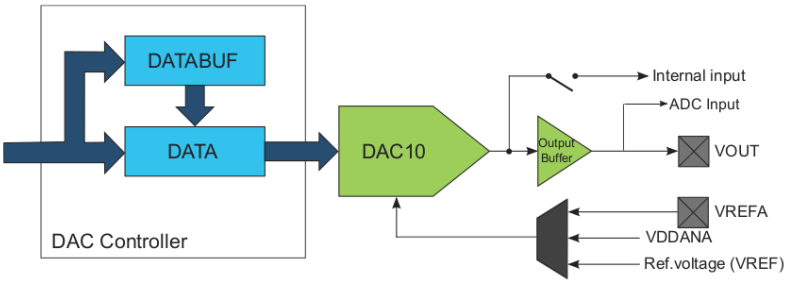

FIGURE 25 – Description diagramme bloc DAC

Les données à convertir transitent sur le bus APB BRIDGE C et doivent être dans le registre `DATA` ou `DATABUFER`. Ils correspondent respectivement à la valeur actuellement convertie et à la prochaine valeur convertie. Dans notre cas nous n'utiliserons que le registre `DATA`. Sa valeur est ensuite transmise à l'unité de conversion 10 bits dont la tension de référence, tension maximum, peut être sélectionnée parmi plusieurs sources. Le résultat sorte finalement sur une sortie interne, un buffer capable de débiter un courant plus important. Comme pour les autres périphériques un certain nombre de registres sont à disposition pour configurer ses nombreuses fonctionnalités.

Configuration du registre CTRLA

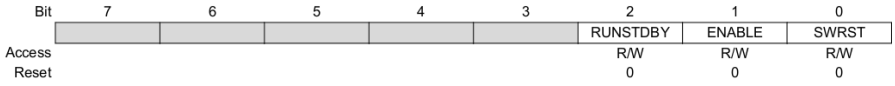

FIGURE 26 – Description registre DAC CTRLA

— **SWRST** - Software Reset - Ce champ permet de remettre les registres du DAC dans leur état initial. Nous ne souhaitons pas faire cela, on écrira dans le registre **0 <<0**

— **ENABLE** - Enable DAC Controller - Ce champ permet d'activer le Dac il faut donc écrire **1 <<1** dans le registre.

— **RUNSTDBY** - Run in Standby - Ce champ permet d'activer ou de désactiver le buffer du DAC et mode veille. Dans notre cas cela a pet d'importance se bit peut être mis à 1 le laisser activé dans tous les cas : **1 <<2**.

Configuration du registre CTRLB

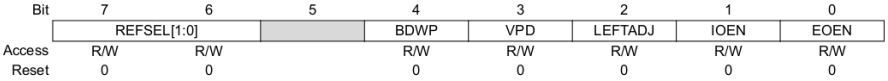

FIGURE 27 – Description registre DAC CTRLB

— **EOEN** - External Output Enable - Ce champ permet d'activer ou de désactiver la sortie du buffer. Nous ne souhaitons appliquer la tension sur le pin Vout, on écrit dans le registre **1 <<0**

— **IOEN** - Internal Output Enable - La sortie interne n'est pas utilisée on écrit **0 <<1** dans le registre.

— **LEFTADJ** Left-Adjusted Data - La taille du registre `DATA` est de 16 bits mais seul les 10 premiers sont utilisés. Ce champs permet de sélectionner les bits en partant de la droite ou de la gauche. Dans notre cas les poids faibles sont à droite on choisira l'option d'alignement à droite. On écrit dans le registre **0 <<2**

— **VPD** - Voltage Pump Disabled - Cette option permet d'économiser de l'énergie. Sa gestion et conservée comme automatique par l'écriture de **1 <<3** dans le registre.

— **BDWP** - Bypass DATABUF Write Protection - Le registre DATABUF n'est pas utilisé, il n'est donc pas nécessaire de le protéger, on écrit **1 <<4** dans le registre.

— **REFSEL[1 :0]** - Reference Selection - Ce registre permet de sélectionner la source de tension de référence. Nous prendrons l'alimentation destinée aux circuits analogiques en écrivant **0x1 <<6** dans le registre.

Initialisation du registre DATA

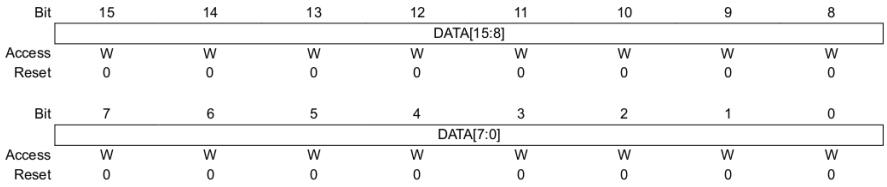

FIGURE 28 – Description registre DAC CTRLB

Ce registre contient la valeur 10 bits à convertir dont l'alignement a été sélectionné avec le registre `LEFTADJ`. Pour tester la bonne configuration se champs pourra être chargé à 1023/2. Un tension de $3,3/3V$ devrait pouvoir être observée.

La configuration du DAC traduite en langage C donne le résultat suivant :

```
void config_DAC(void){
  Dac *ptr_DAC = DAC ;

  ptr_DAC -> CTRLA.reg =  DAC_CTRLA_RUNSTDBY   |
                          DAC_CTRLA_ENABLE     |
                          DAC_CTRLA_SWRST_NO ;

  ptr_DAC -> CTRLB.reg =  DAC_CTRLB_REFSEL_AVCC |
                          DAC_CTRLB_BDWP_NO     |
                          DAC_CTRLB_VPD_NO      |
                          DAC_CTRLB_LEFTADJ_NO  |
                          DAC_CTRLB_IOEN_NO     |
                          DAC_CTRLB_EOEN ;
  /* Initialisation pour test */
  ptr_DAC -> DATA.reg = 1024/2;
  /* DATA.reg valeur 10bits max */
}
```

De nouveau l'utilisation des structures de l'ASF facilité grandement le travail. Cela améliore aussi la compréhension et donc la modification par un tiers.

7 Implantation de la commande PWM et modélisation du moteur

Le cahier des charges requiert une commande de type PWM pour le moteur. Cette étape consiste à mettre en œuvre ce type de commande en utilisant un Timer. Le moteur sera quant à lui modélisé par un filtre passe-bas de constantes de temps 10ms. Éventuellement, ce filtre passe-bas pourra avoir une fonction d'atténuation, afin d'adapter le niveau de sortie du PWM à celui d'entrée du CAN. Le filtre sera alors réalisé par deux résistances et un condensateur. Selon la configuration du CAN qu'il sera bon d'étudier dans cette étape, il faudra trouver la structure de ce filtre ainsi que les valeurs des composants.

7.1 Génération du signal PWM

7.1.1 Présentation la fonctionnement du TC en mode PWM

Il nous faut ici réaliser une commande de type PWM pour le moteur, représentative du gabarit mis en place. Le principe de la PWM est le suivant :

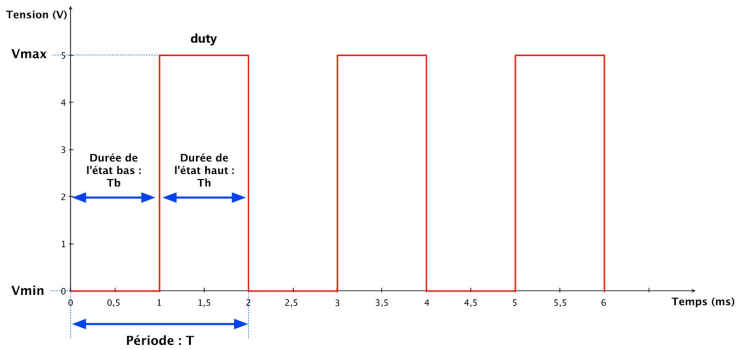

FIGURE 29 – Principe de la PWM

Le signal émis est rectangulaire et varie entre 2 états de tension : Umax et Umin (généralement 0V). La tension moyenne de ce signal, notée Umoy, s'évalue de la façon suivante :

$$U_{moy} = \frac{1}{T}\int_0^T u(t)\,\mathrm{d}t = \frac{1}{T}[U_{max} * t]_0^{T_h} = U_{max}\frac{T_h}{T} \tag{5}$$

La tension moyenne ne dépend donc que du rapport cyclique entre T_h et T. Si T_h est nul, la tension moyenne est nulle. Si $T_h = T$, la tension moyenne est maximale.
En se basant sur ce principe, le but est de transformer l'information de Vconsigne en une variation du rapport cyclique d'un signal envoyé au moteur. Si Vconsigne augmente, le rapport augmente et si Vconsigne atteint VpalierMax alors le rapport est maximal. Cette fonction est réalisable grâce aux timers présent sur la carte. En effet il est possible de programmer un timer en mode MPWM (Match Pulse-Width Modulation). Le fonctionnement de ce mode est le suivant :

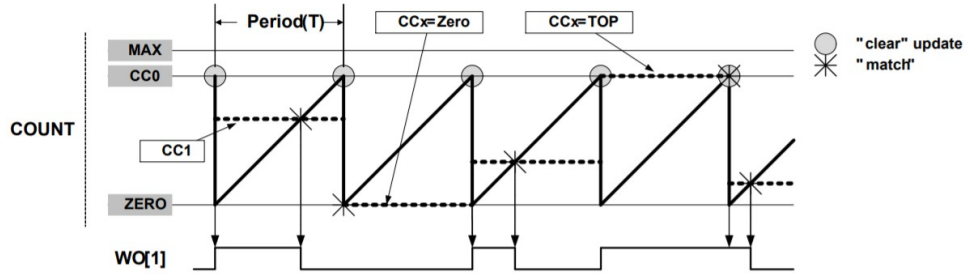

FIGURE 30 – Fonctionnement du mode MPWM

Ce mode permet de créer en sortie un signal de période définie par le rapport entre CC1 et CC0. Donc le rapport cyclique est contrôlable. Tant que COUNT a une valeur inférieur a CC1, WO[1] est à l'état haut puis il passe a l'état bas lorsque COUNT est entre CC1 est CC0.
Le but de cette application étant d'avoir une fréquence de 10 kHz, il faut donc diviser l'horloge mère par 800 c'est à dire prendre un CC0 d'une valeur de 799.
Le CC1 ne sera pas constant contrairement à CC0, il faudra l'actualiser à chaque fois que la vitesse imposée est changée soit toutes les microsecondes. La valeur de CC1 sera définie par la formule suivante :

$$CC1 = \frac{Vconsigne * CC0}{VpalierMax} = \frac{Vconsigne * 799}{VpalierMax} \quad (6)$$

On sait que $0<Vconsigne<VpalierMax$. CC1 évoluera donc entre 0 et CC0 ce qui permettra d'avoir un rapport cyclique entre 0 et 100% comme souhaite.

7.1.2 Configuration et programmation

Au niveau, le timer TC7 sera utilisé. En effet celui-ci est déjà activé dans le registre GCLK car l'identifiant de TC7 est lié à celui de TC6 (voir partie 4.2). Il est donc préférable d'utiliser ce timer plutôt que d'en déclarer un nouveau. Les codes reprend en grande partie le principe de TC6 vu précédemment.
La partie sur le registre GCLK n'a pas besoin d'être réécrite car TC6 et TC7 partageant le même identifiant, l'horloge est déclarée pour les 2 timers lors de l'initialisation de TC6.

Configuration du Power Manager(PM)
L'activation du timer TC7 auprès du Power Manager sans altérer les composants déjà présents. On ajout le code suivant dans le fonction config_projet() qui est créé pour le TC6 au début :

```
1 void config_PM(void){
2       Pm *ptr_PM = PM;
3
4       ptr_PM -> APBCMASK.reg |=           PM_APBCMASK_TC7;
5 }
6
```

Configuration des ports d'E/S parallèles
D'après la datasheet SAMD21, la sortie WO[1] du TC7 est disponible sur les pin PA21, PB01 et PB23. Pour des raisons pratiques vis-à-vis de la disposition des pins sur la carte, la pin PB01 est

choisie. WO[1] est donc récupéré en sortie sur la pin 1 du port B en multiplexage E. On ajout le code suivant dans la méthode config_PORT() :

```
void config_PORT(void){
  /* PWM */
  ptr_port -> Group[1].DIRSET.reg =  PORT_PB01;
  ptr_port -> Group[1].PMUX[1/2].reg = PORT_PMUX_PMUXO_E;
  ptr_port -> Group[1].PINCFG[1].reg = PORT_PINCFG_DRVSTR_NO |
                                       PORT_PINCFG_PULLEN_NO |
                                       PORT_PINCFG_INEN_NO |
                                       PORT_PINCFG_PMUXEN;
}
```

A noter que 1 est impair, il faut donc spécifier le multiplexage E dans la partie impaire du registre PMUX0. D'où l'utilisation de la constante PORT_PMUX_PMUXO_E.

Configuration du périphérique Timer Counter 7

Une nouvelle fonction `config_PWM()` est créé dans le programme.
La configuration du registre CTRLA du TC7, reprenant les mêmes éléments que pour TC6, sauf au niveau du mode sélectionné. Ici le mode est le MPWM et mis en place par la constante TC_CTRLA_WAVEGEN_MPWM.

```
void config_PWM (void){
  Tc *ptr_TC = TC7 ;

  ptr_TC -> COUNT16.CTRLA.reg = TC_CTRLA_PRESCSYNC_PRESC |
                                TC_CTRLA_NORUNSTDBY |
                                TC_CTRLA_PRESCALER_DIV1 |
                                TC_CTRLA_WAVEGEN_MPWM |
                                TC_CTRLA_MODE_COUNT16 |
                                TC_CTRLA_NOENABLE |
                                TC_CTRLA_NOSWRST;
  //......
}
```

La configuration des registres CTRLC et CTRLBCLR du TC7, reprenant la même logique que pour TC6, d'où la configuration similaire.

```
    ptr_TC -> COUNT16.CTRLC.reg = TC_CTRLC_NOCPTEN0 |
                                  TC_CTRLC_NOCPTEN1 |
                                  TC_CTRLC_NOINVEN0 |
                                  TC_CTRLC_NOINVEN1;

    ptr_TC -> COUNT16.CTRLBCLR.reg = TC_CTRLBCLR_CMD_NONE |
                                     TC_CTRLBCLR_ONESHOT |
                                     TC_CTRLBCLR_NODIR;
```

La configuration des 2 champs CC0 et CC1. CC1 est initialisé à 400 et variera par la suite. CC0 est affecté de la constante TC_CC0_Val, constante à la valeur 799 comme exprimé précédemment.

```
    ptr_TC -> COUNT16.CC[0].reg = TC_CC0_Val; // 10kHz
    ptr_TC -> COUNT16.CC[1].reg = 400; // duty 50%
```

L'activation du timer TC7 en fin d'initialisation.

```
    ptr_TC -> COUNT16.CTRLA.reg |= TC_CTRLA_ENABLE;
```

Le timer TC7 est à présent défini, il faut maintenant mettre en place la fonction permettant de faire évoluer le palier CC1 en fonction de *duty* (Vconsigne). Cette fonction est la suivante :

```
void write_PWM (uint16_t duty){
  Tc *ptr_TC = TC7 ;

  ptr_TC -> COUNT16.CC[1].reg = (duty * TC_CC0_Val) / V_MAX;
}
```

CC1 est défini sur [0 : TC_TC0_Val], c'est-à-dire [0 : 799]. V_MAX est la constante de la valeur 5000.

7.2 Dimensionnement du filtre modélisant le moteur

7.2.1 Calculs pour un filtre passe bas

Pour ce projet, le moteur est représenté par une un filtre passe-bas ainsi qu'un étage d'atténuation (pour effectuer des ajustements sur les plages de tension) composés de résistances et de condensateurs. La structure utilisée est la suivante :

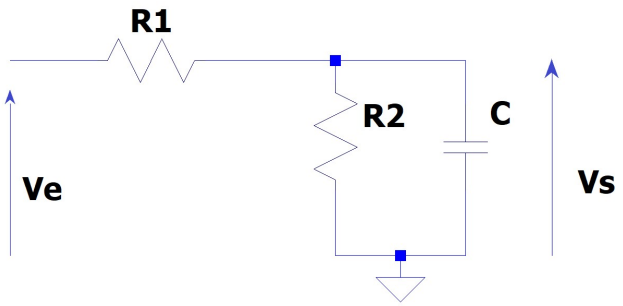

FIGURE 31 – Schéma du filtre passe-bas

La fonction de transfert est déterminée par l'équation suivante :

$$H(p) = \frac{G}{1 + \tau p} \tag{7}$$

Avec

$$G = \frac{V_s}{V_e} = \frac{R_2}{R_1 + R_2} \tag{8}$$

$$\tau = (R_1 // R_2)C = \frac{R_1 R_2}{R_1 + R_2} C \tag{9}$$

Il nous à présent dimensionner les composants. Le choix du filtre doit permettre d'avoir une tension de sortie égale à $\frac{3,3}{2}$V c'est-à-dire 1,65V. Pour cela, il faut travailler avec REFSEL à 0x2 pour avoir INTVCC1, c'est-à-dire $\frac{1}{2}$ de V_{DDANA} mais tant que cela reste [valeur comparée]<2,7V.

REFSEL[3:0]	Name	Description
0x0	INT1V	1.0V voltage reference
0x1	INTVCC0	1/1.48 VDDANA
0x2	INTVCC1	1/2 VDDANA (only for VDDANA > 2.0V)
0x3	VREFA	External reference
0x4	VREFB	External reference
0x5-0xF		Reserved

FIGURE 32 – Tensions de références disponibles pour le CAN

Pour obtenir le signal désiré en sortie du filtre, il faut déterminer le gain. Celui-ci est $V_{DDANA}/2$ soit $G = \frac{1}{2}$. De plus, la constante de temps de ce filtre est prise à 10 ms d'après le cahier des charges. Donc les composants reposent sur 2 conditions exprimées dans l'étape :

— Constante de temps du filtre passe bas de 10ms
— Gain de 0,5

Selon l'équation (8) et (9), on peut obtenir :

$$\frac{R_1 R_2}{R_1 + R_2} C = 10ms \tag{10}$$

$$\frac{R_2}{R_1 + R_2} = \frac{1}{2} \tag{11}$$

D'après l'équation (11), on suppose que $R_1 = R_2 = R$. On peut obtenir alors

$$\frac{R^2}{2R} C = \frac{R}{2} C = 10ms \tag{12}$$

Le choix de C étant limité dans les valeurs de composants existants, il est défini à 10 μF. Donc la valeur de R est 2,2 kΩ.

7.2.2 Test du filtre passe bas

Le circuit peut à présent être réalisé sur la planche à pain. Pour vérifier la constante de temps, nous utilisons l'oscilloscope pour observer la sortie du filtre. Le résultat est présenté dans la figure ci-dessous :

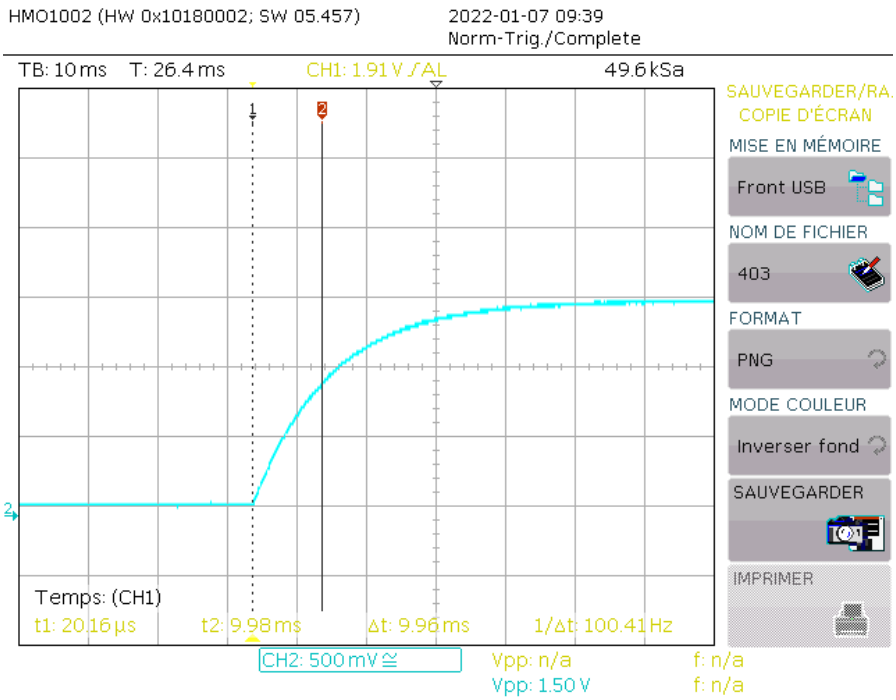

FIGURE 33 – Test de la sotie du filtre passe bas

D'après le résultat du test, la constante de temps est environ 10 ms qui vérifie le cahier des charges. Une fois le montage effectué, il suffit de relier les cartes ensemble ensuite de faire la simulation.

7.3 Résultats de l'implantation