projet
document

Louison GOUY Yiying WEI

15 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

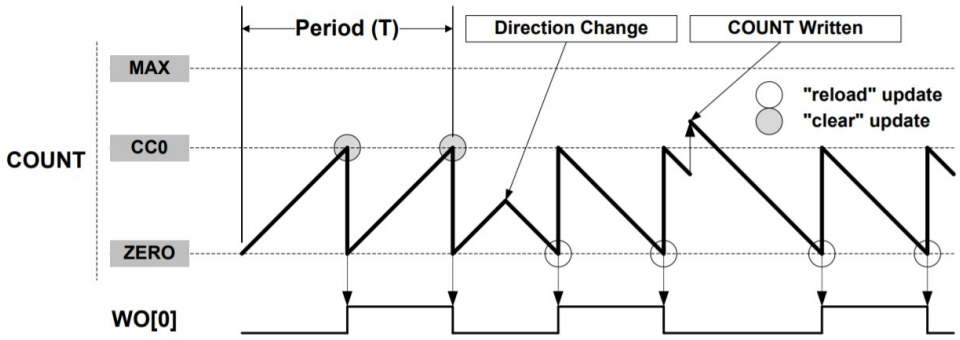

FIGURE 2 – Fonctionnement timer configuré en MFRQ

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.

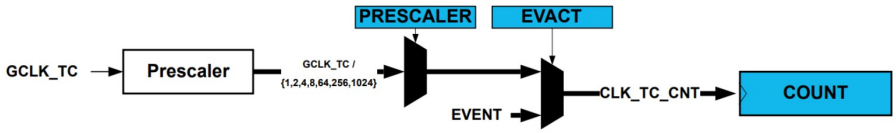

FIGURE 3 – Configuration du compteur TC

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.

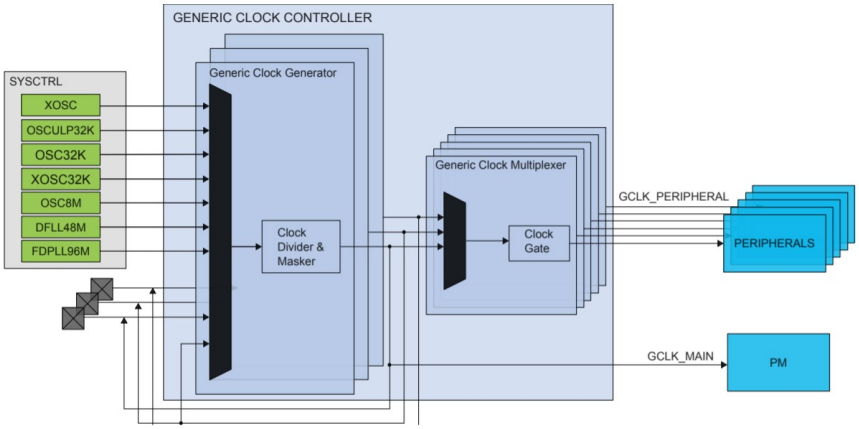

FIGURE 4 – Fonctionnement du Generic Clock Controller

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 :

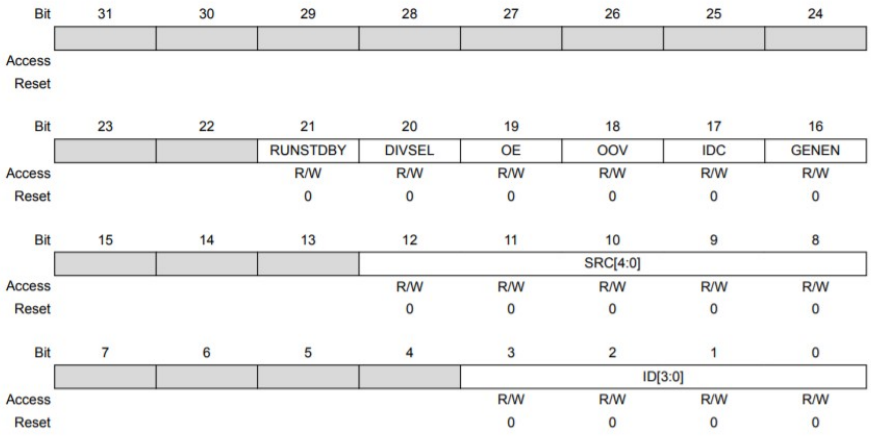

FIGURE 5 – Configuration des différents bits du GENCTRL

— **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 :

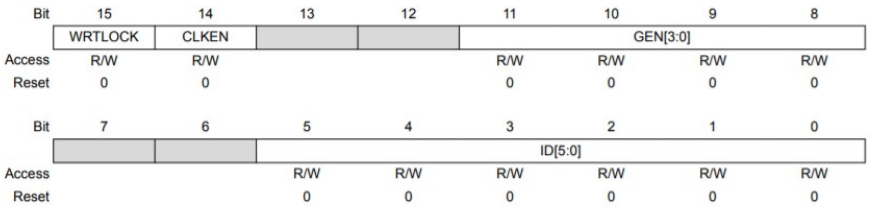

FIGURE 6 – Configuration des différents bits du CLKCTRL

— **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**.

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.

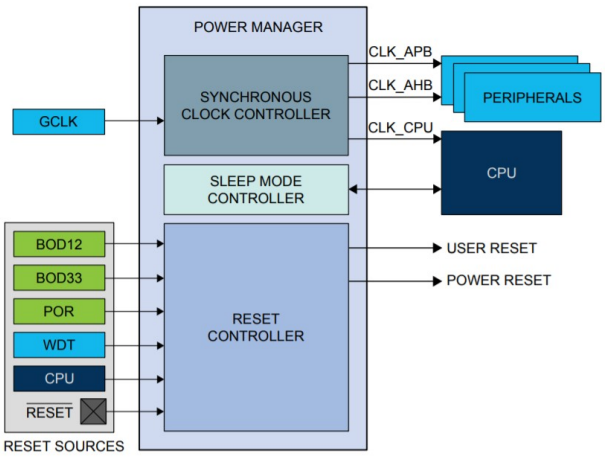

FIGURE 7 – Structure du Power Manager

On programme le "Synchronous Clock Controller" en entré CLK, en sortie 3 horloges par les 3 bus interes du Microprocesseur : APBA, APBB et APBC.

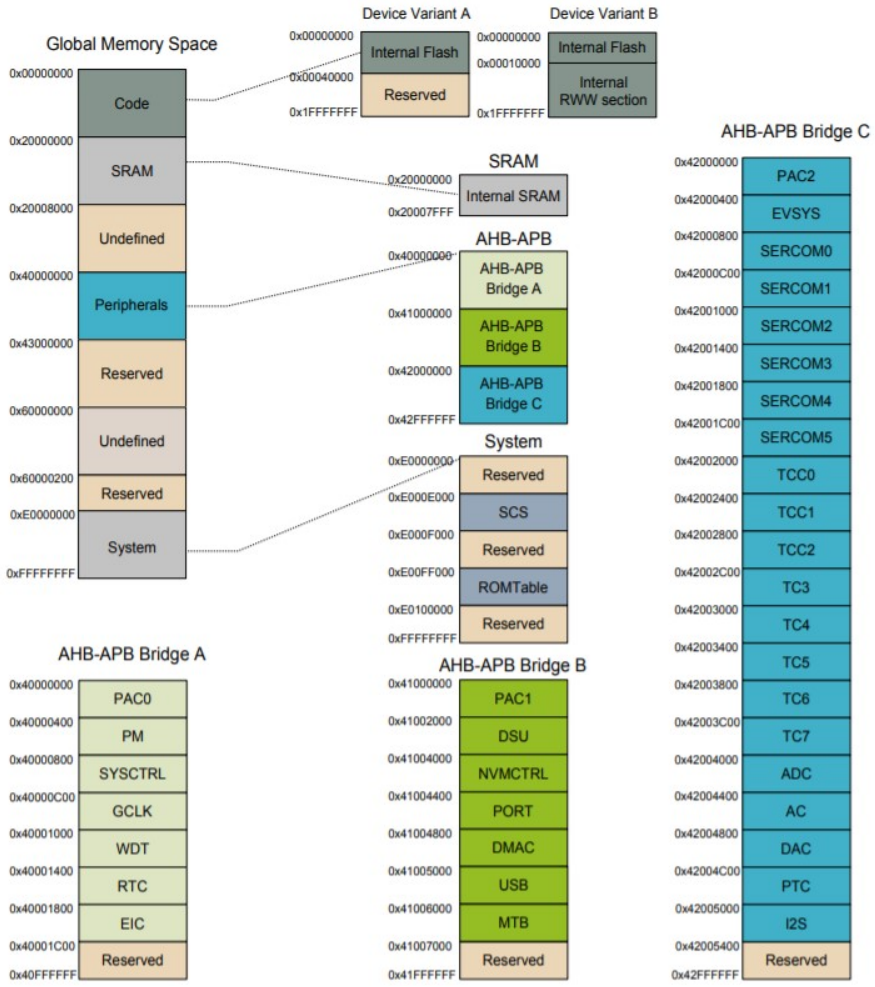

FIGURE 8 – Cartographie des produits Atmel SAM D21

D'après la cartographie des produits, 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 :

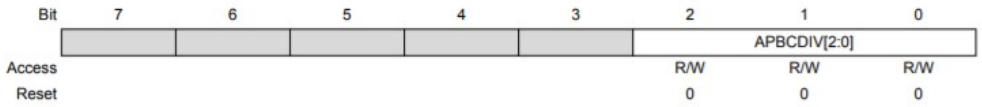

FIGURE 9 – Configuration des différents bits du APBCSEL

— **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.

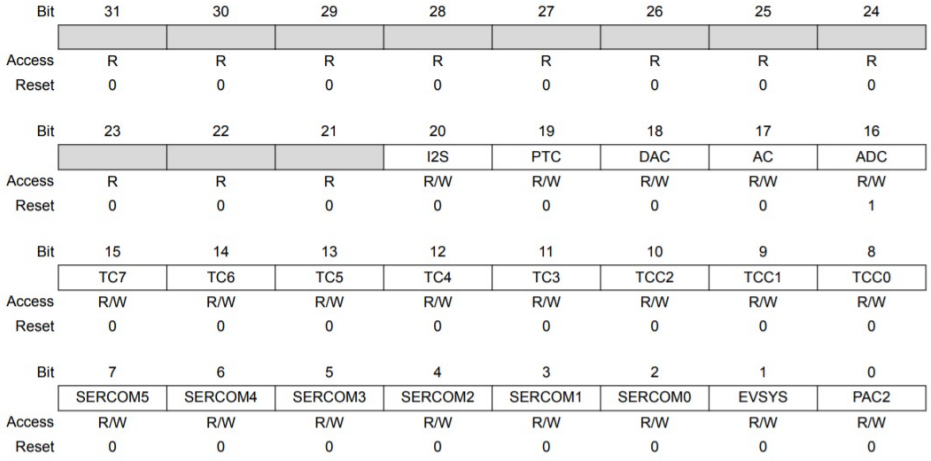

FIGURE 10 – Configuration des différents bits du APBCMASK

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 :

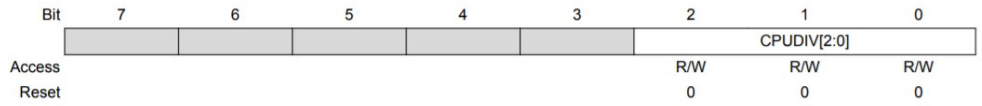

FIGURE 11 – Configuration des différents bits du CPUSEL

— **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**.

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 12 – 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 13 – 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 14 – 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 15 – 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**.

4.2.4 Configuration du périphérique

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 :

Bit	15	14	13	12	11	10	9	8
			PRESCSYNC[1:0]		RUNSTDBY	PRESCALER[2:0]		
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
		WAVEGEN[1:0]			MODE[1:0]		ENABLE	SWRST
Access		R/W	R/W		R/W	R/W	R/W	R/W
Reset		0	0		0	0	0	0

FIGURE 16 – 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 :

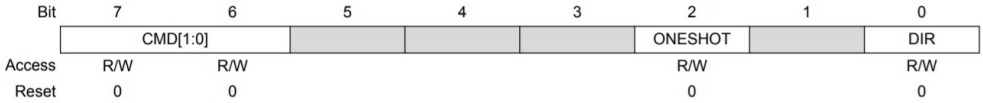

FIGURE 17 – Configuration des différents bits du CTRLBCLR

— **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 :

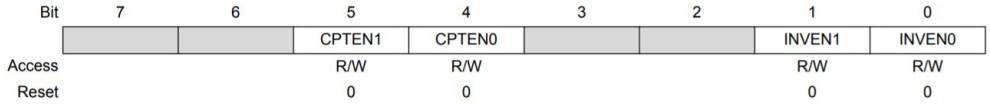

FIGURE 18 – Configuration des différents bits du CTRLC

— **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.

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 :

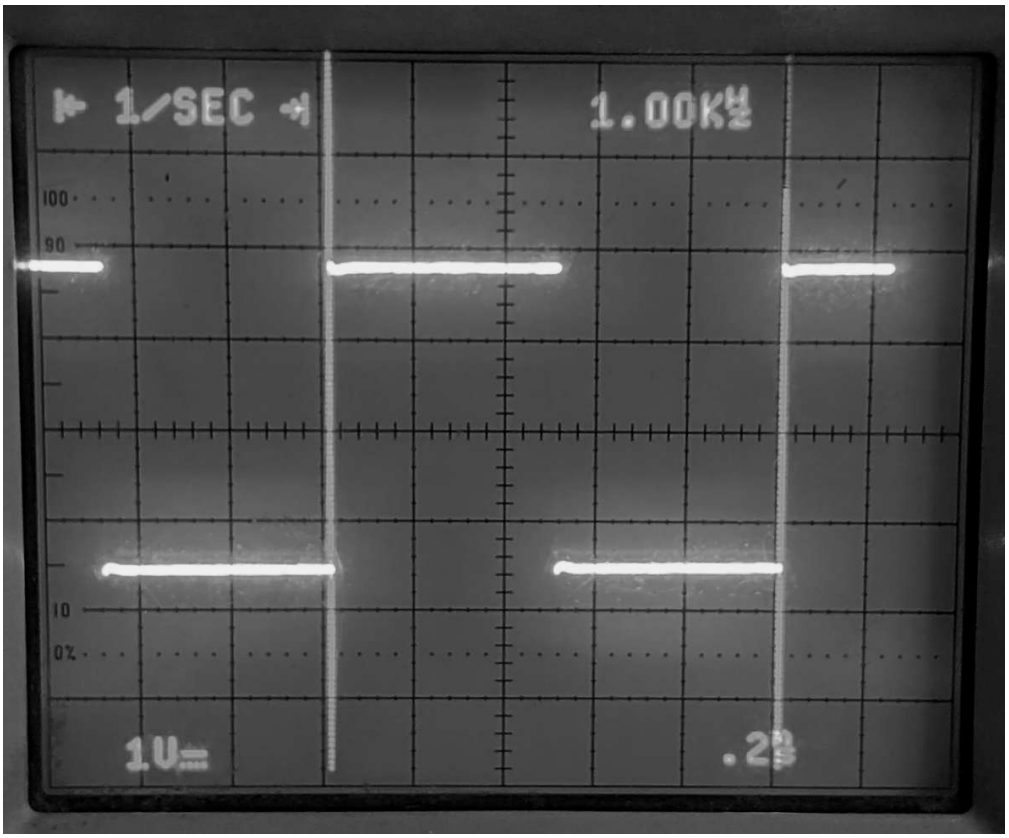

FIGURE 19 – Signal de sortie du TC6

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.

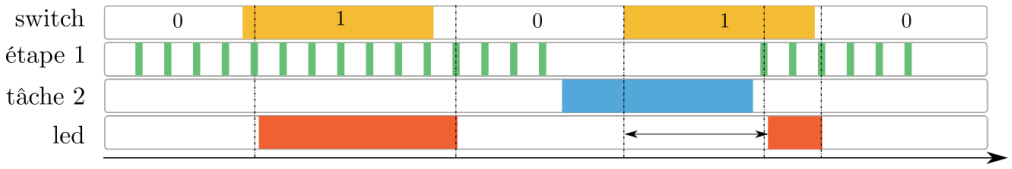

FIGURE 20 – Ordonnancement tâches avec polling

La figure ci-dessus représente l'exécution d'un programme comme celui de l'étape 1 auquel une seconde tâche a été ajoutée (tâche 2). Aucune échelle précise n'est choisi le but étant une simple présentation fonctionnelle. 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 ale 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 : event-trigger) offre également une meilleur réactivité. Des ressources pourront alors être économisées en baissant la fréquence ne fonctionnement. [**?**]

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.