

# **projet** document

Louison GOUY      Yiying WEI

7 janvier 2022

name

abstract

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

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 tâche. [?]

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 se 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 considéreront 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'entrée 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 particulière 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 défini. 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` défini 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<sup>1</sup>. 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.

---

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

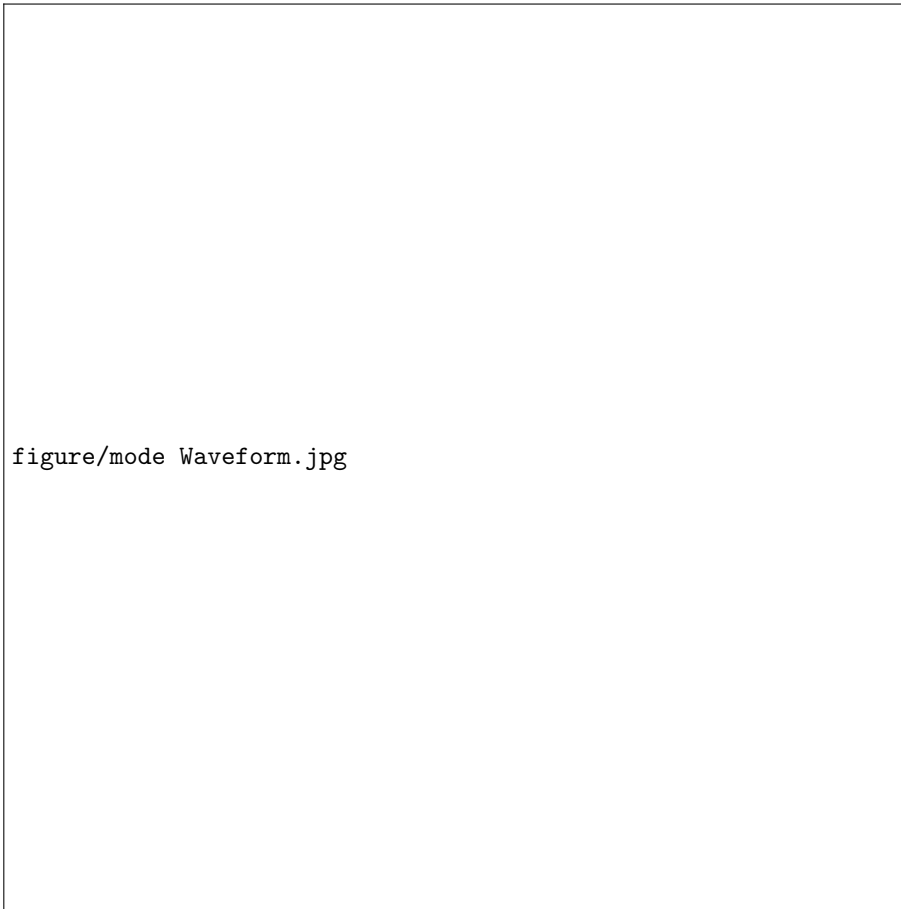
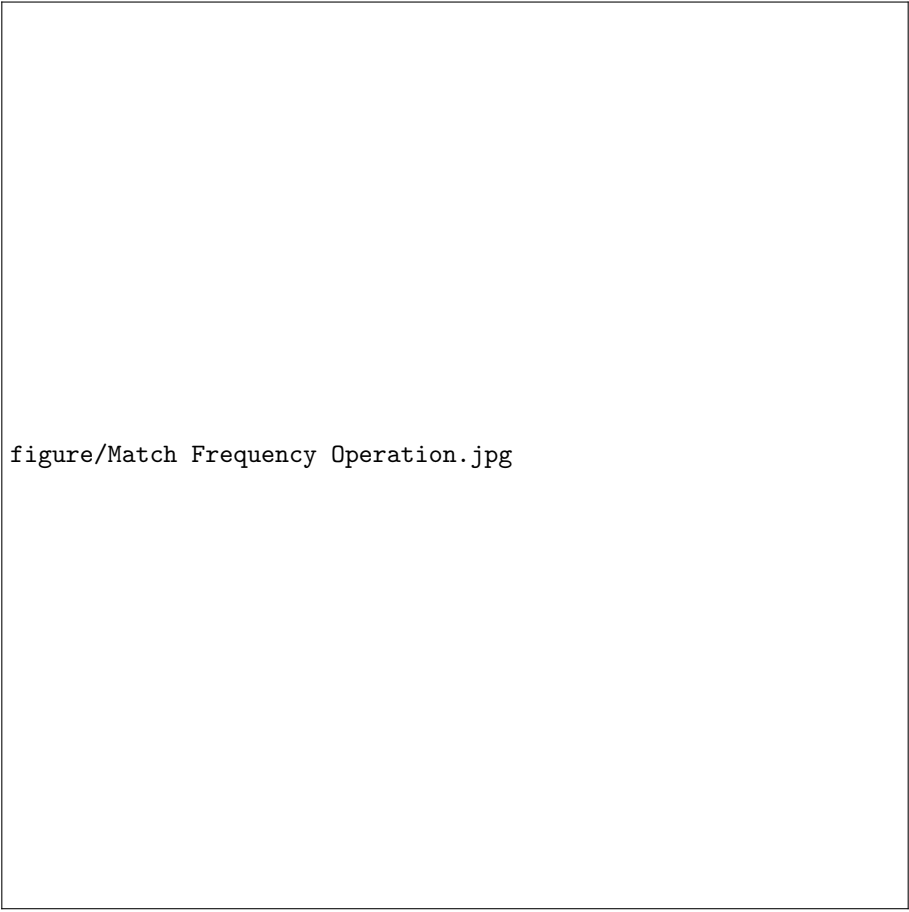


FIGURE 1 – Mode de fonctionnement en WAVEFORM pour les compteurs

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.



figure/Match Frequency Operation.jpg

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 vigilant 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.

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.



figure/prescaler.jpg

FIGURE 3 – Configuration du compteur TC

$$T_{GCLK.TC} = N * T_{CLK.TC.CNT} = T_{CLK.TC.CNT} \quad (1)$$

Donc

$$f_{GCLK.TC} = f_{CLK.TC.CNT} \quad (2)$$

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

$$f_{WO[0]} = \frac{f_{CLK.TC.CNT}}{2 * (CC0 + 1)} \quad (3)$$

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

$$CC0 = \frac{f_{GCLK.TC}}{2 * f_{WO[0]}} - 1 = 3999 \quad (4)$$

La valeur chargée dans le registre CC0 sera donc 3999.

## 4.2 Configuration 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 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.

On remarque que le Generic Clock Controller est divisé en deux parties. D'une part le Generic Clock Controller est configuré par le registre GENCTRL. D'autre part le Generic Clock Multi-plexer 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 :

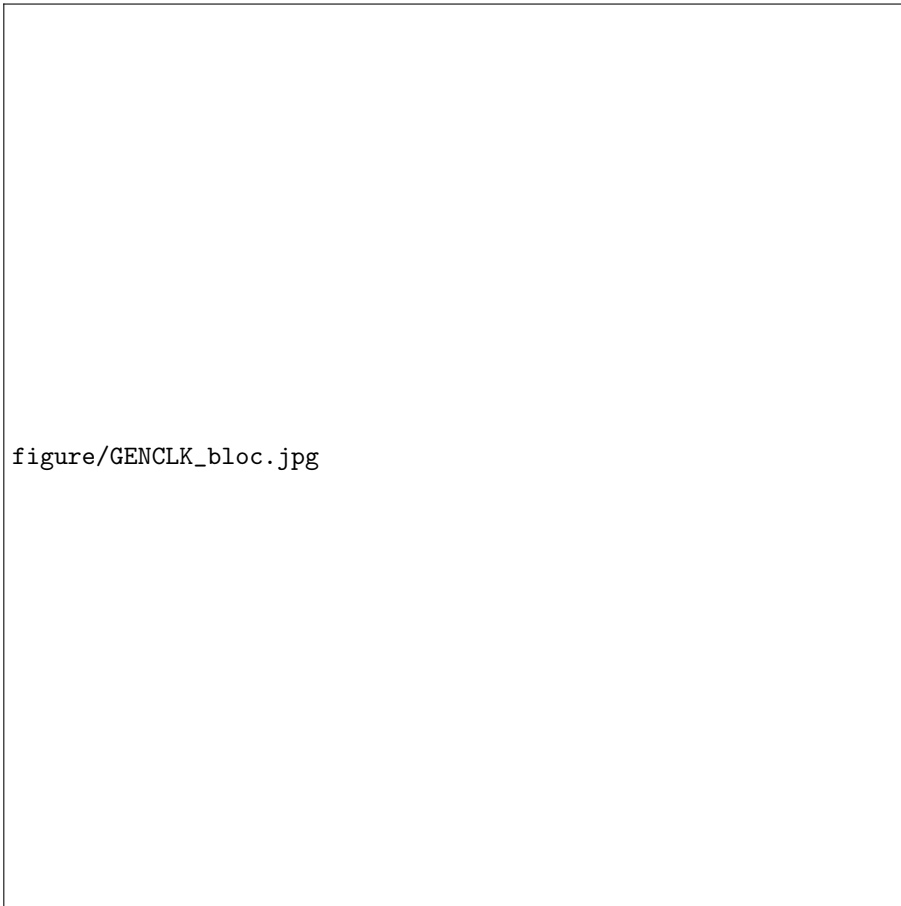
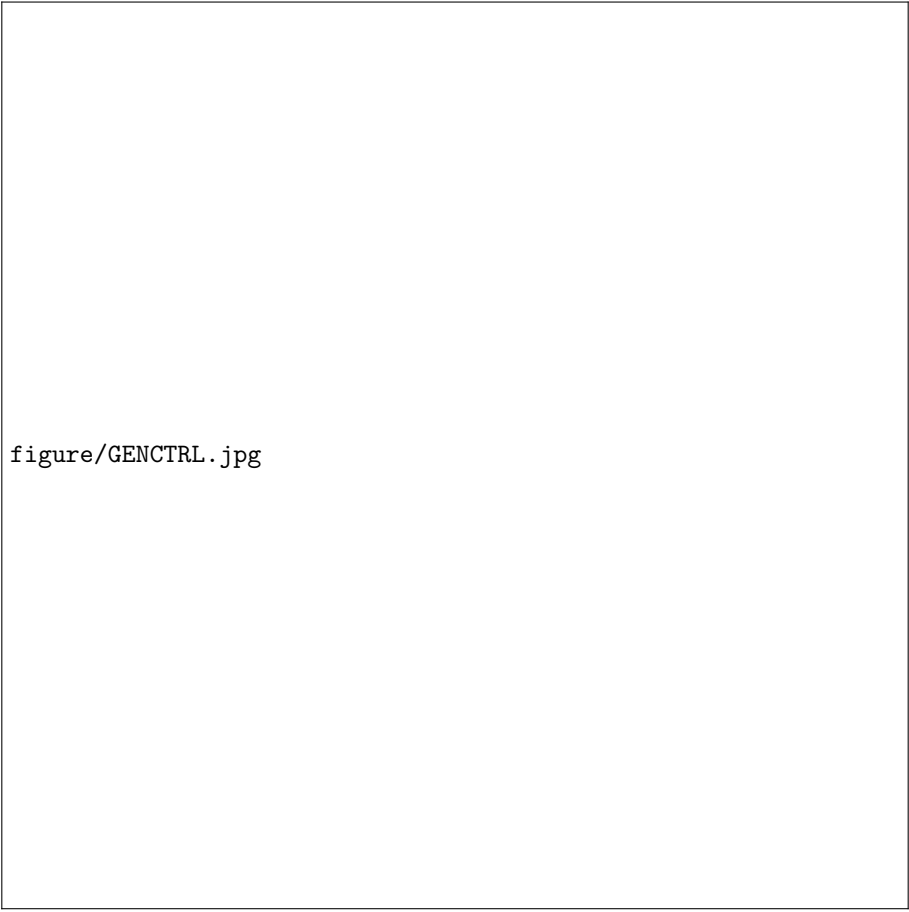


FIGURE 4 – Fonctionnement du Generic Clock Controller



figure/GENCTRL.jpg

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  $0 \ll 21$  dans ce champ.
- **DIVSEL** : Définit le facteur de division de l'horloge. Nous ne voulons pas la diviser donc il faut mettre la valeur  $0 \ll 20$  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  $0 \ll 19$  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  $0 \ll 18$ .
- **IDC** : Définit du rapport cyclique en cas de division impaire. Dans notre cas, il faut mettre la valeur  $0 \ll 17$  dans ce champ.
- **GENEN** : Validation ou non du générateur d'horloge. Nous voulons l'activer donc il faut mettre la valeur  $1 \ll 16$  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 parmi les 9 générateurs décrit précédemment. Le détail de Generic Clock Control (CLKCTRL) est donné dans la figure ci-dessous :

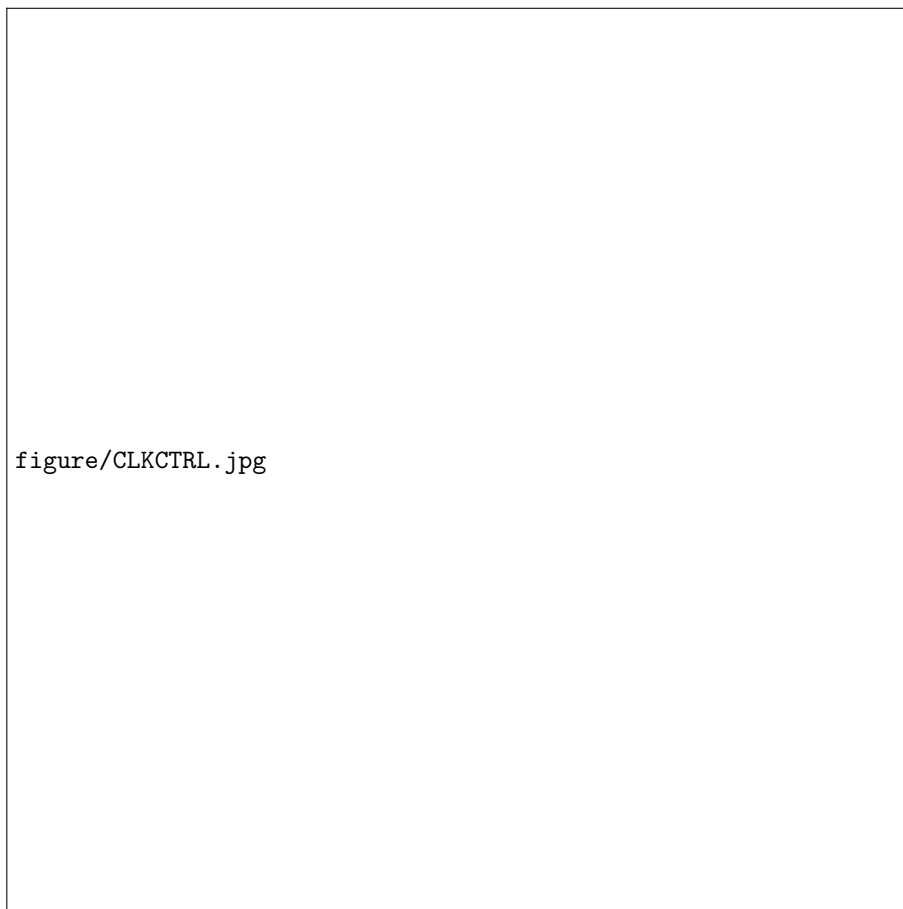


FIGURE 6 – Configuration des différents bits du CLKCTRL

#### 4.2.2 Configuration du PM

#### 4.2.3 Configuration du PORT

#### 4.2.4 Configuration du TC6